计算机学院专业必修课

计算机组成

多周期MIPS：数据通路/控制
时序逻辑：状态机

高小鹏

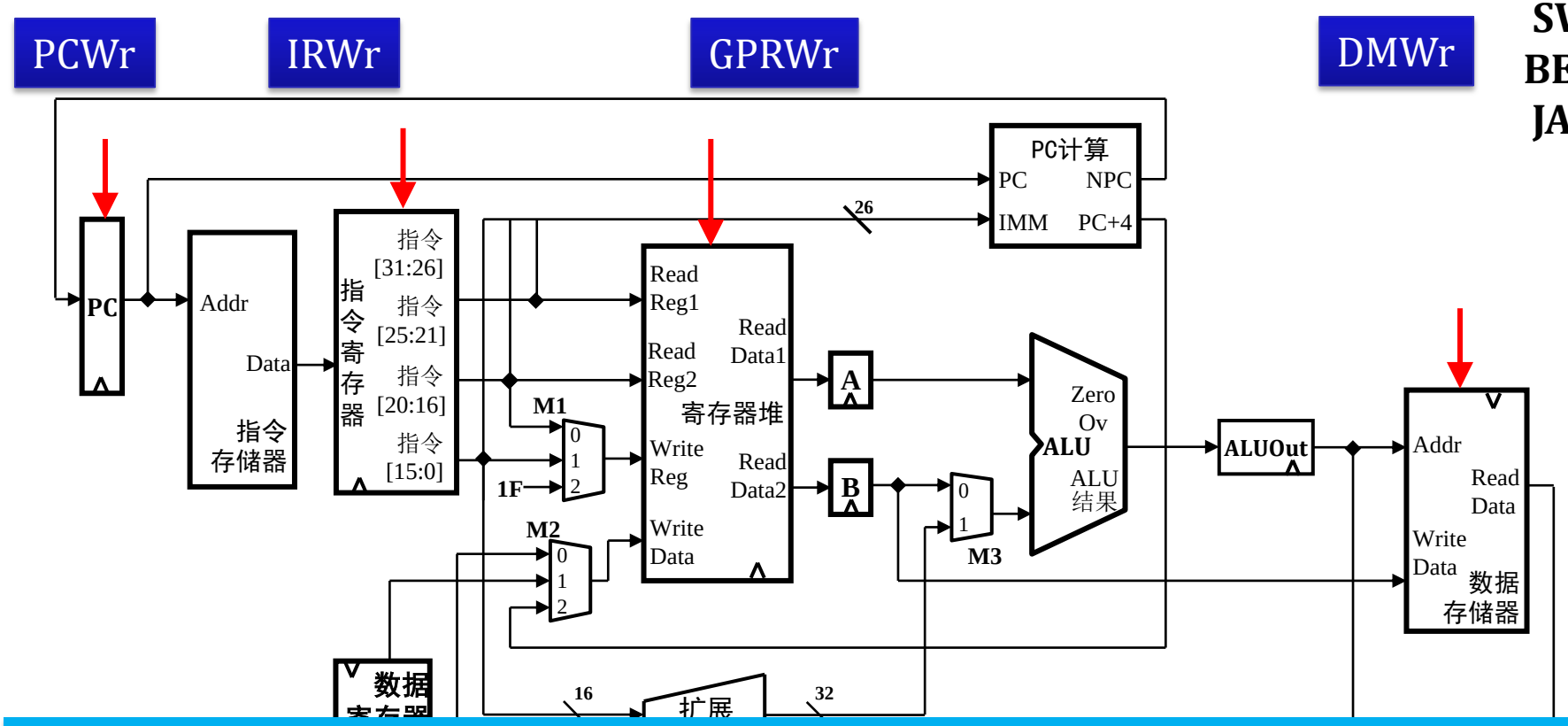
北京航空航天大学计算机学院
系统结构研究所

提纲

- 内容主要取材
  - 数字设计和计算机体系结构（第3章，第7章）
- 多周期数据通路控制信号分析
- 多周期控制器状态机构造
- 多周期性能分析

多周期数据通路控制信号：寄存器写使能

LW
ADDU
SUBU
ORI
LUI
SW
BEQ
JAL

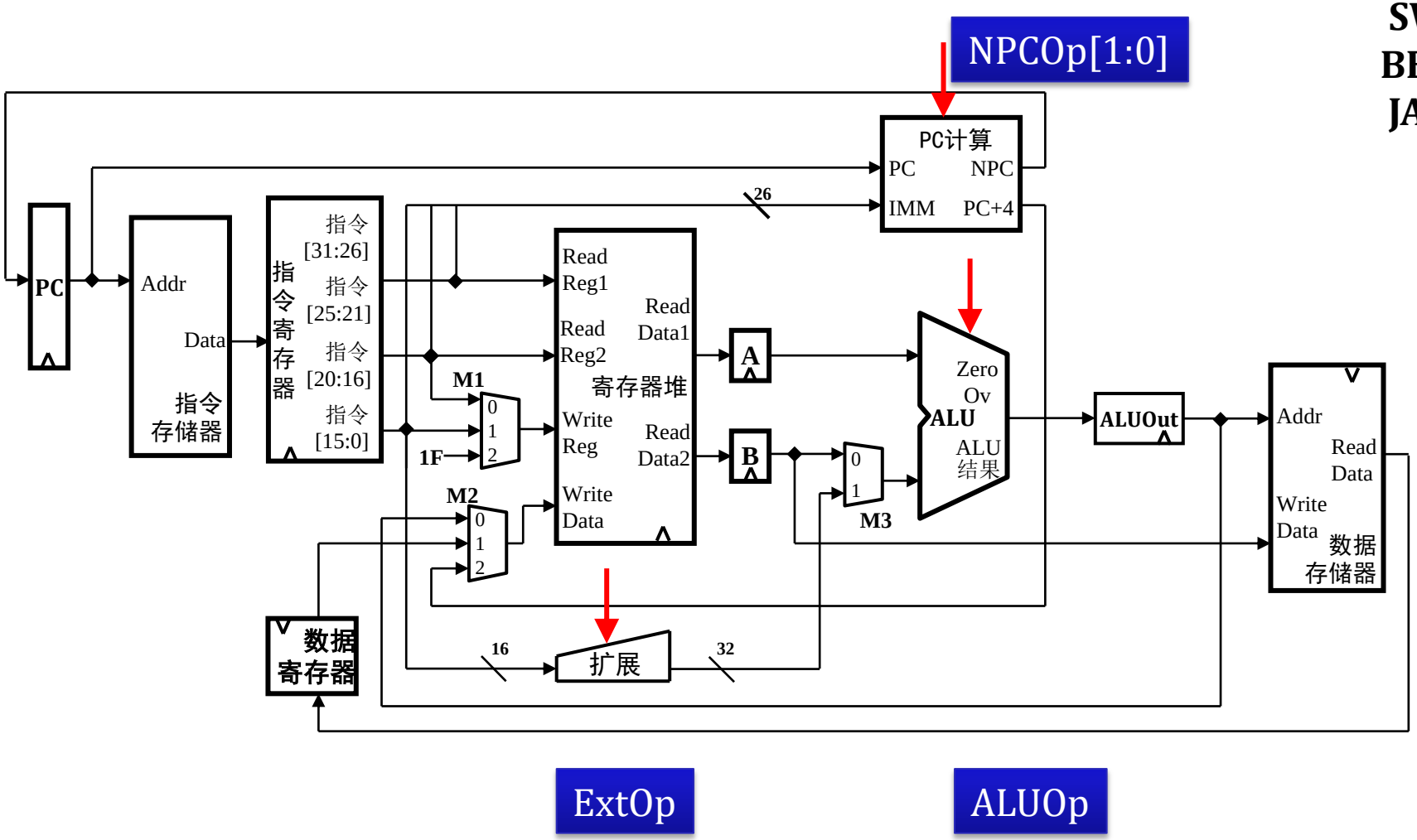


写使能信号

1：允许写入；0：禁止写入

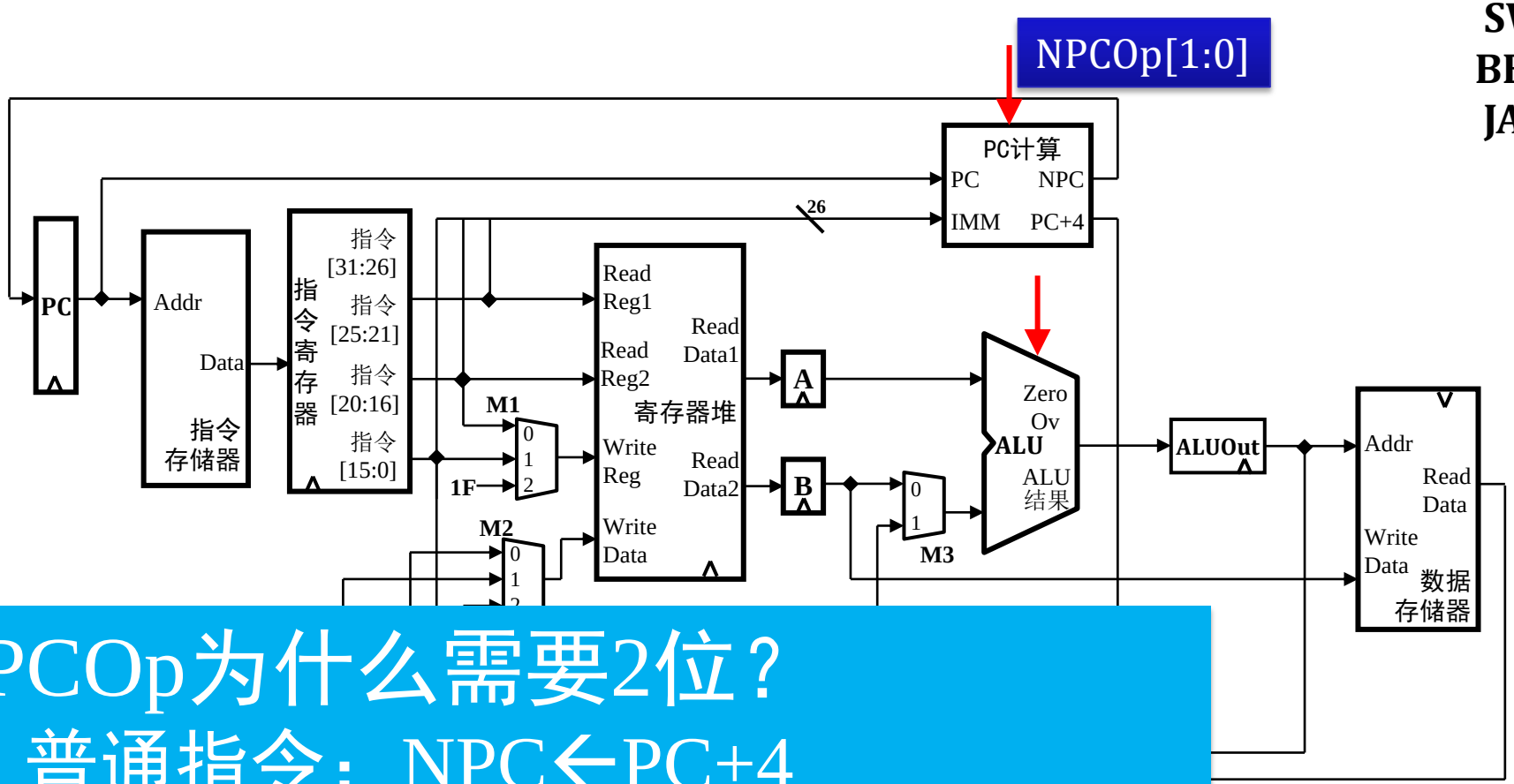
多周期数据通路控制信号：操作选择

LW
ADDU
SUBU
ORI
LUI
SW
BEQ
JAL



多周期数据通路控制信号：操作选择

LW
ADDU
SUBU
ORI
LUI
SW
BEQ
JAL



NPCOp为什么需要2位？

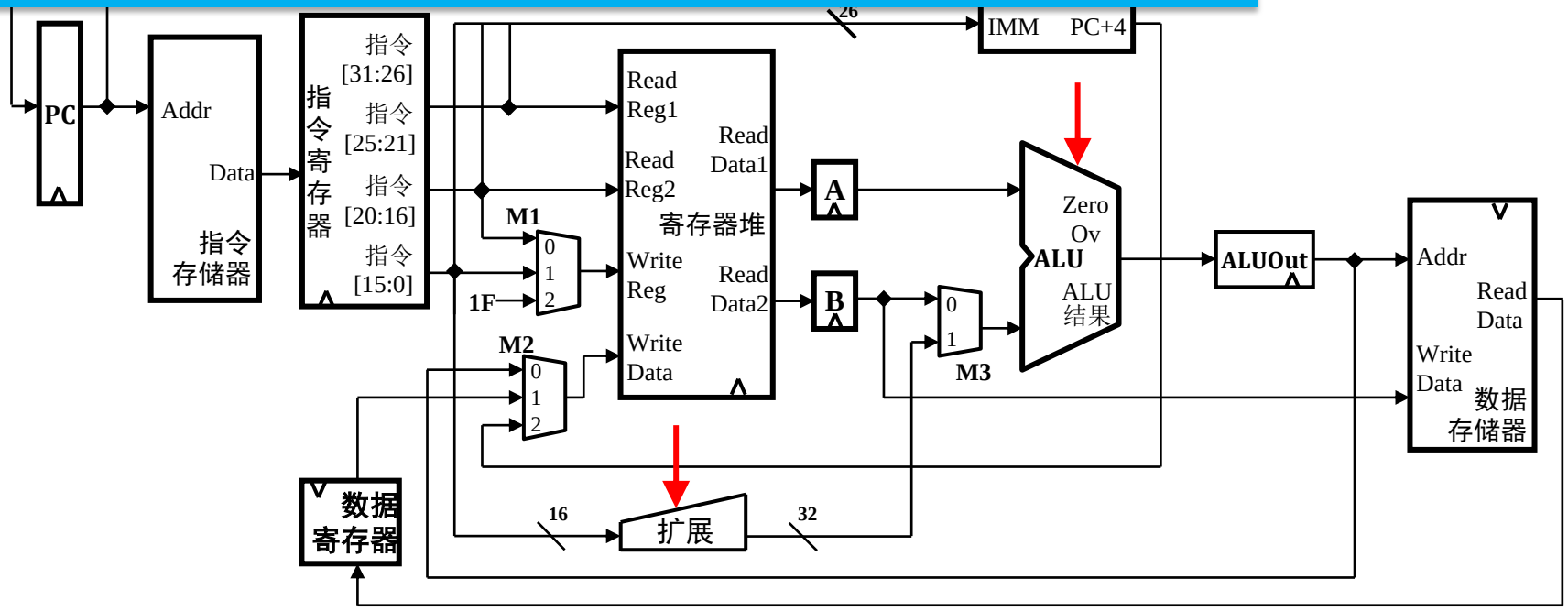
- 普通指令：NPC←PC+4
- BEQ：NPC←PC+sign_ext(imm16)
- JAL：NPC←$PC_{31..28}$ || imm26 || 00

多周期数据通路控制信号：操作选择

LW
ADDU
SUBU
ORI
LUI
SW
BEQ
JAL

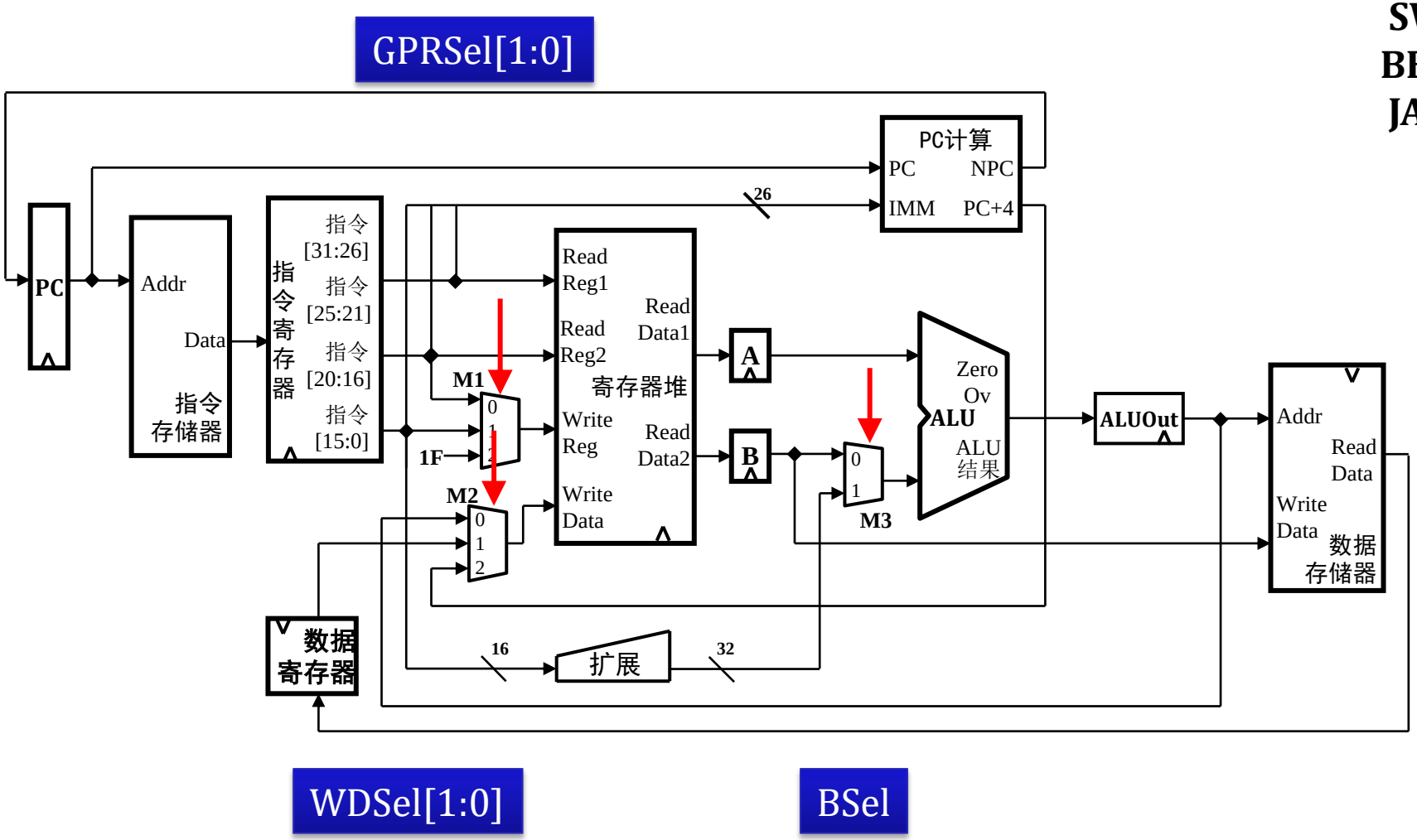
ALUOp为什么需要2位？

- 无符号加、符号加、无符号减、或



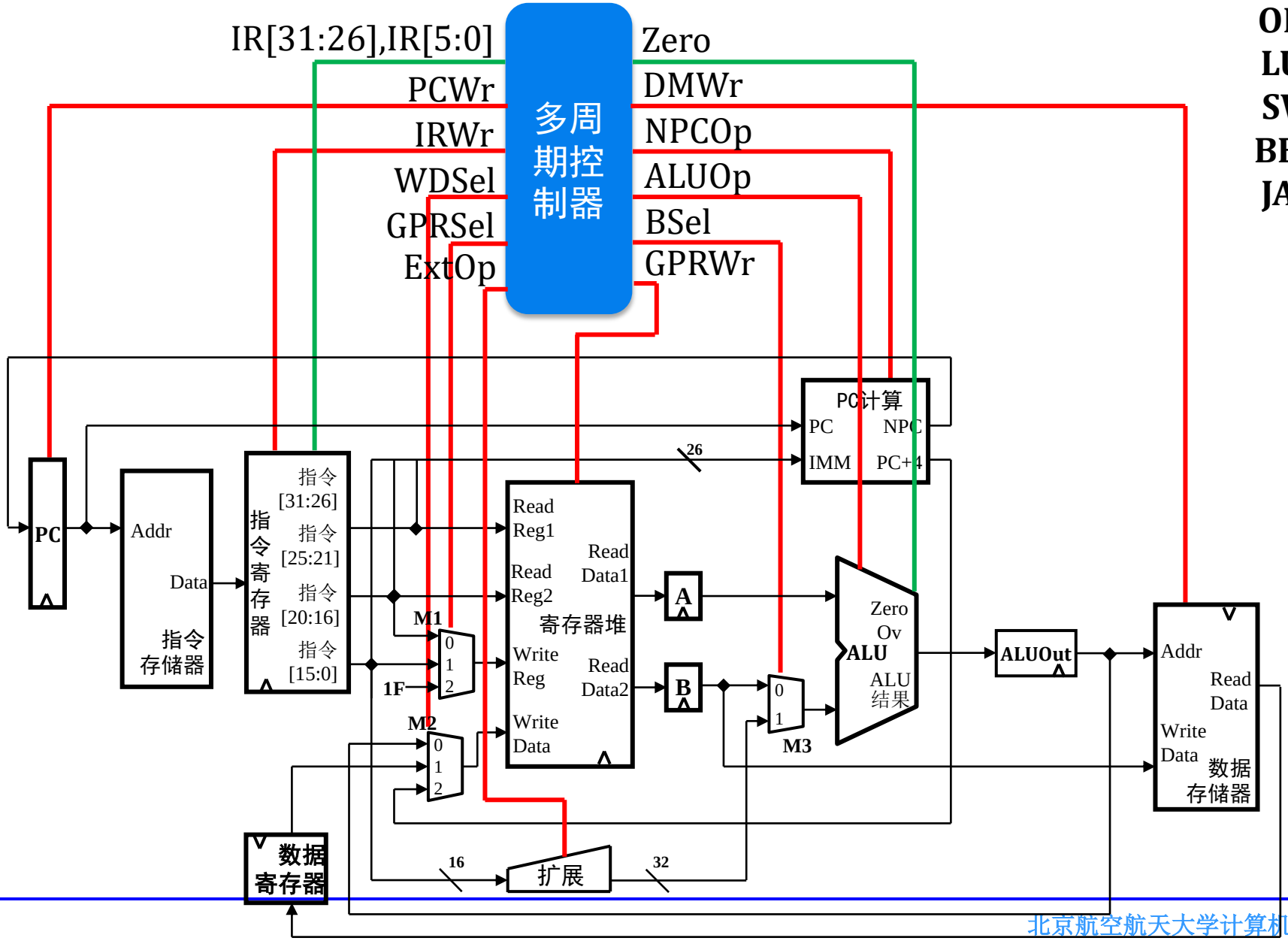
多周期数据通路控制信号：多路选择

LW
ADDU
SUBU
ORI
LUI
SW
BEQ
JAL



多周期数据通路及控制器

LW
ADDU
SUBU
ORI
LUI
SW
BEQ
JAL



提纲

- 内容主要取材
  - 数字设计和计算机体系结构（第3章，第7章）
- 多周期数据通路控制信号分析
- 多周期控制器状态机构造
- 多周期性能分析

多周期控制器设计

- 数据通路：5阶段(IF、DCD/RF、EXE、MEM、WB)
  - 特点：每条指令由若3～5个时钟周期完成
- 控制器：FSM＋输出逻辑
  - FSM至少应包括5个状态：分别对应数据通路的5个阶段
- FSM输入
  - Op/Funct：Instr[31:26]/Instr[5:0]
  - Zero
  - Reset，Clk
- 输出逻辑
  - 指令执行在特定阶段处于某种状态
  - Op/Funct、FSM、Zero

FSM构造过程

- LW，SW，ADDU，SUBU，ORI，LUI，BEQ，JAL

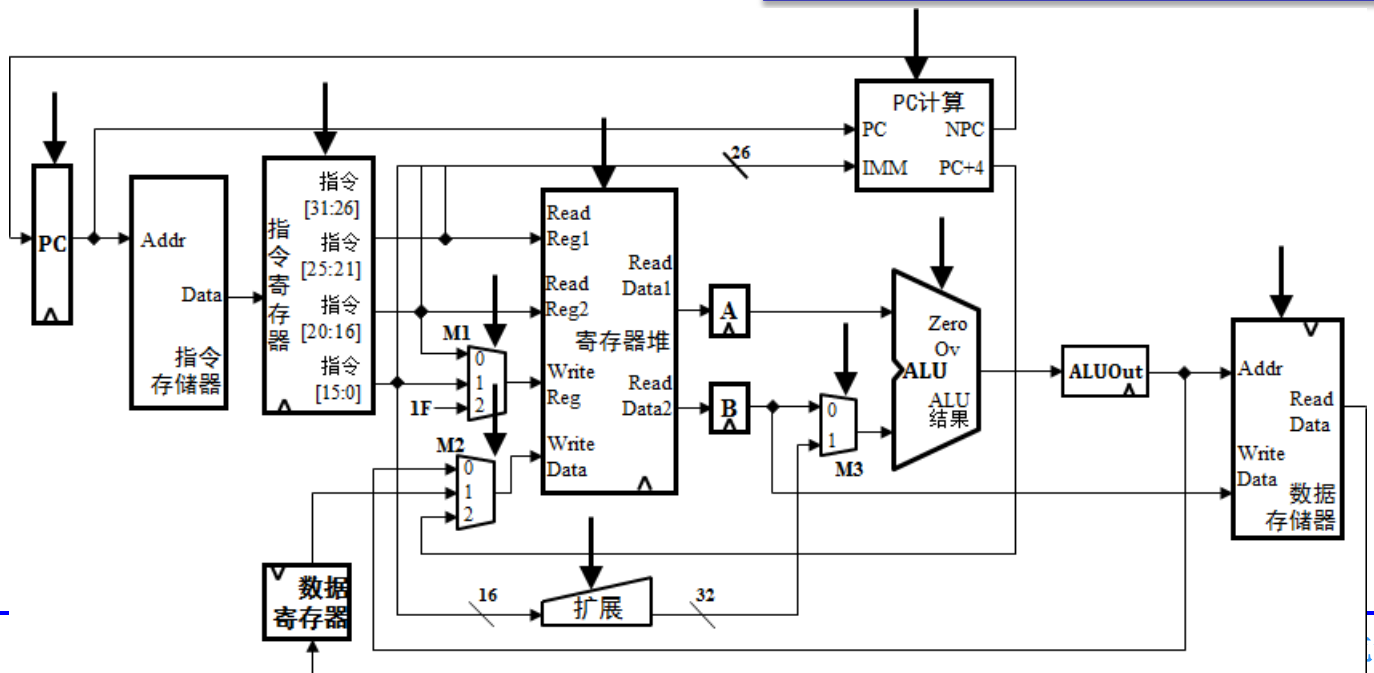
31–26	25–21	20–16	15–0
LW 100011	base	rt	offset
6	5	5	16

```
Addr ← sign_extend(offset) + GPR[base]
memword ← Memory[Addr]
GPR[rt] ← memword
PC ← PC + 4
```

RTL

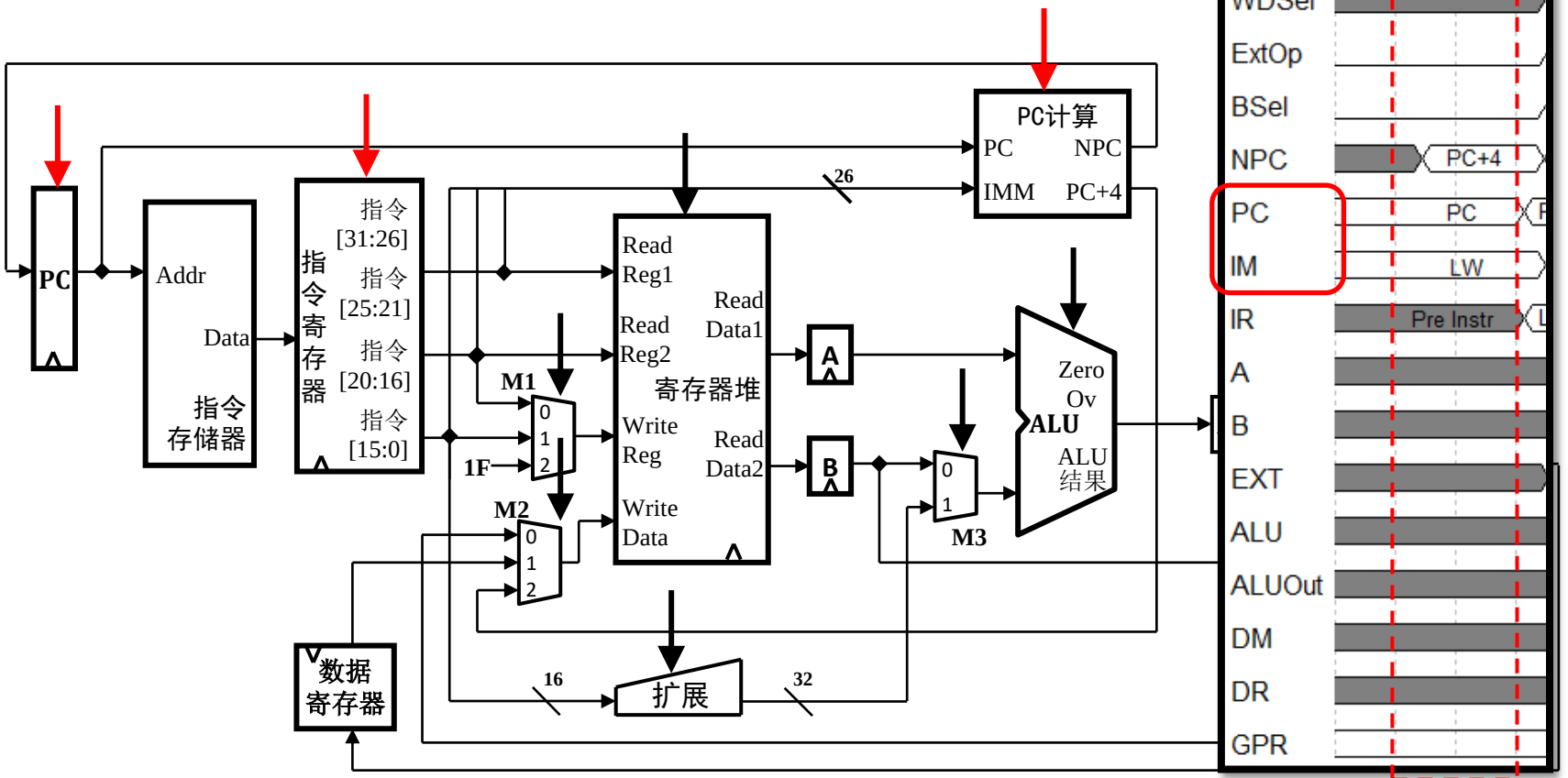
R[rt]←MEM[R[rs]+sign_ext(imm16)]

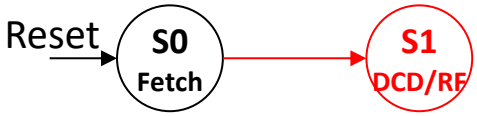
PC←PC+4



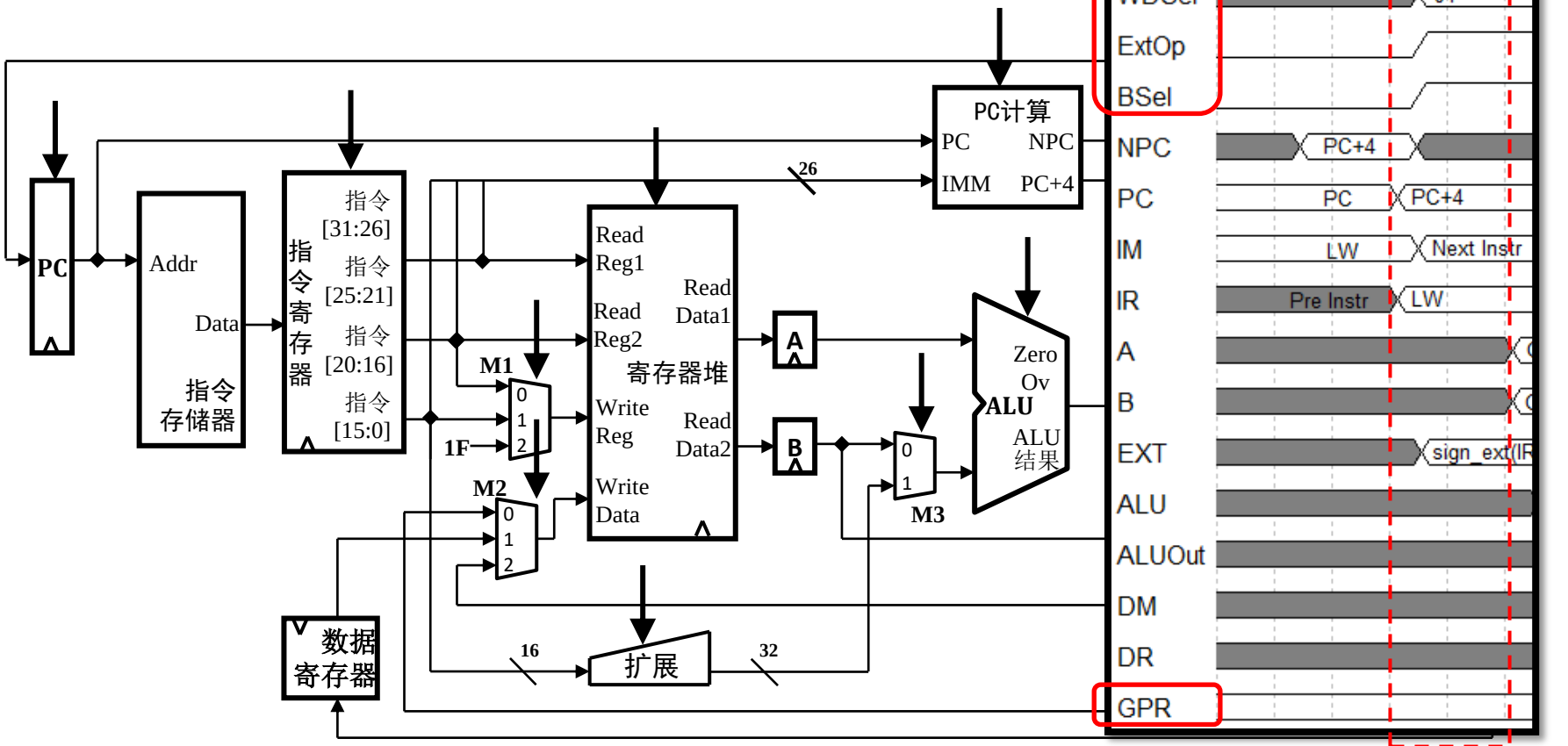
Reset → S0 Fetch

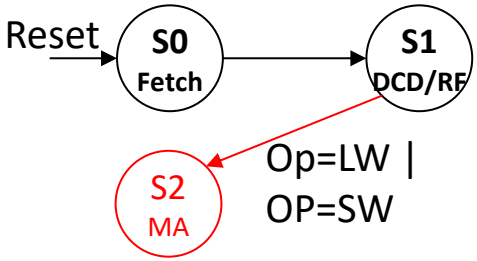
	Op	Funct	S0	S1	S2	S	S9
PCWr			1				
NPCOp			PC+4				
IRWr			1				
GPRWr			0				
DMWr			0				
ALUOp			X				
GPRSel			X				
WDSel			X				
ExtOp			X				
BSel			X				



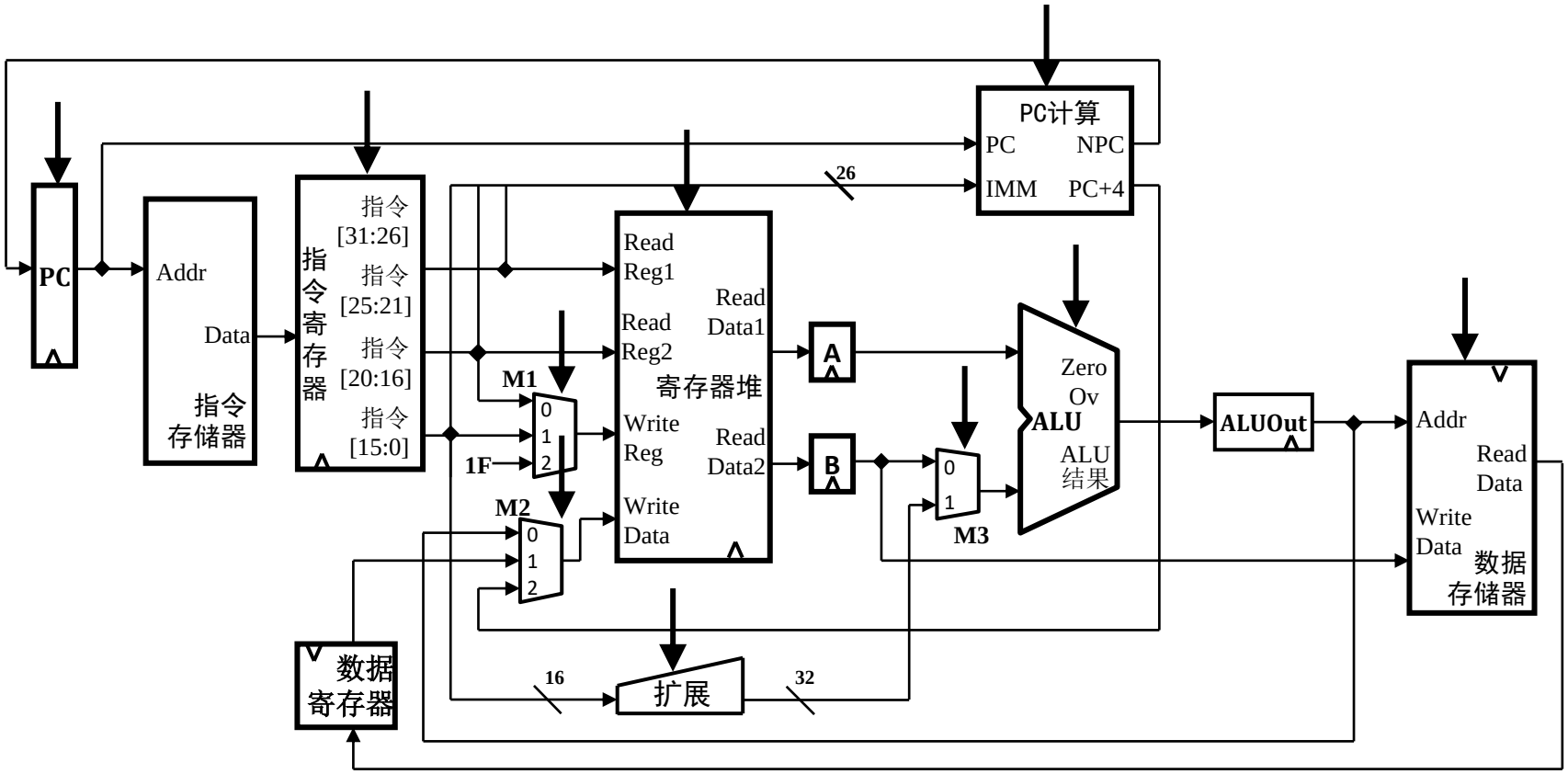


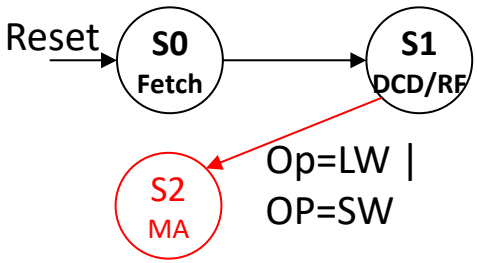
	Op	Funct	S0	S1	S
PCWr			1	0	
NPCOp			PC+4	X	
IRWr			1	0	
GPRWr			0	0	
DMWr			0	0	
ALUOp			X	add	
GPRSel			X	00	
WDSel			X	01	
ExtOp			X	SE	
BSel			X	1	



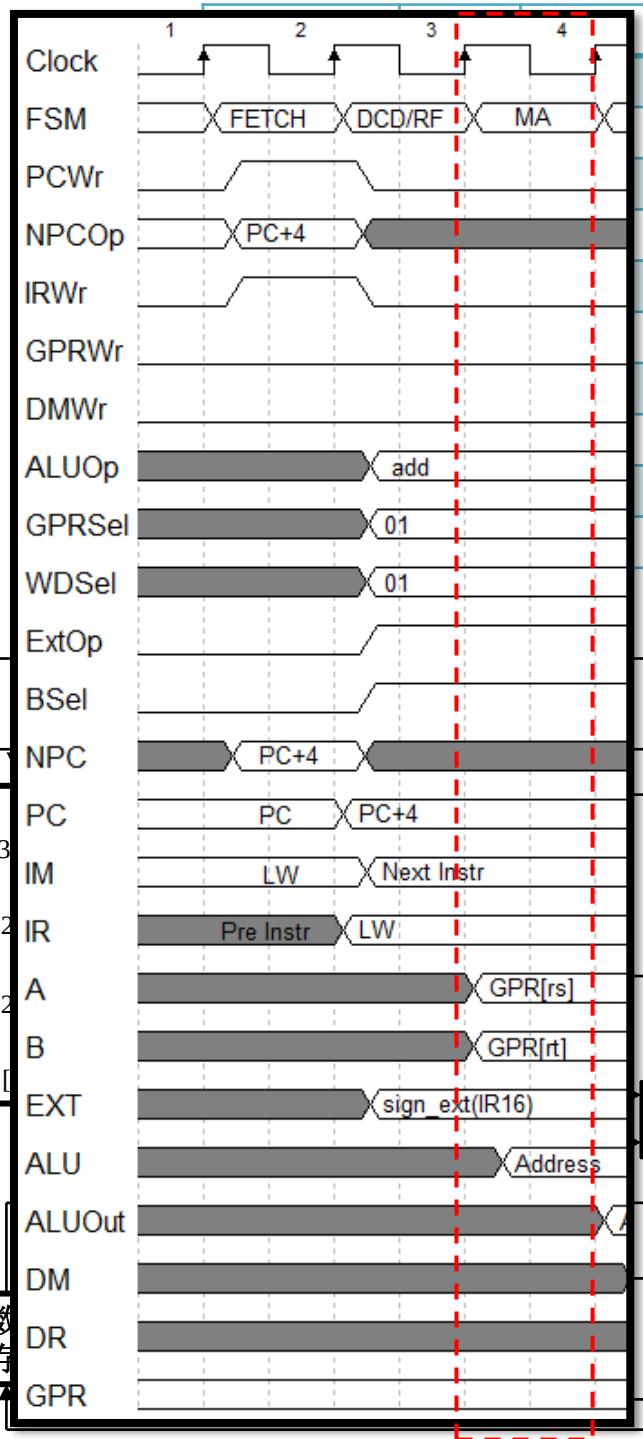


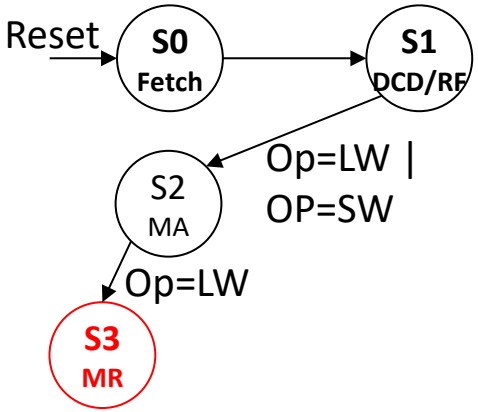
	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0	0							
NPCOp			PC+4	X	X							
IRWr			1	0	0							
GPRWr			0	0	0							
DMWr			0	0	0							
ALUOp			X	add	add							
GPRSel			X	00	00							
WDSel			X	01	01							
ExtOp			X	SE	SE							
BSel			X	1	1							



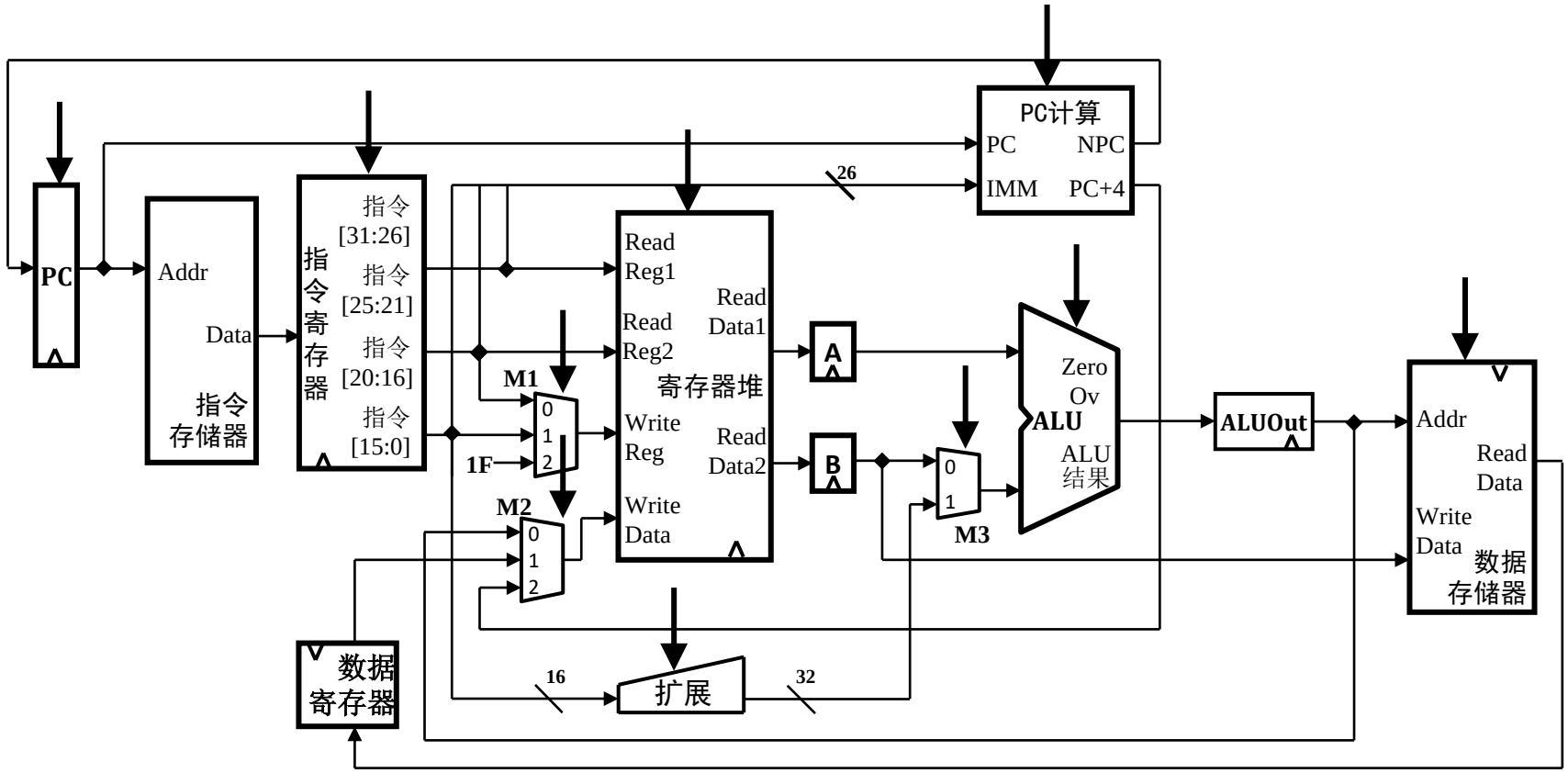


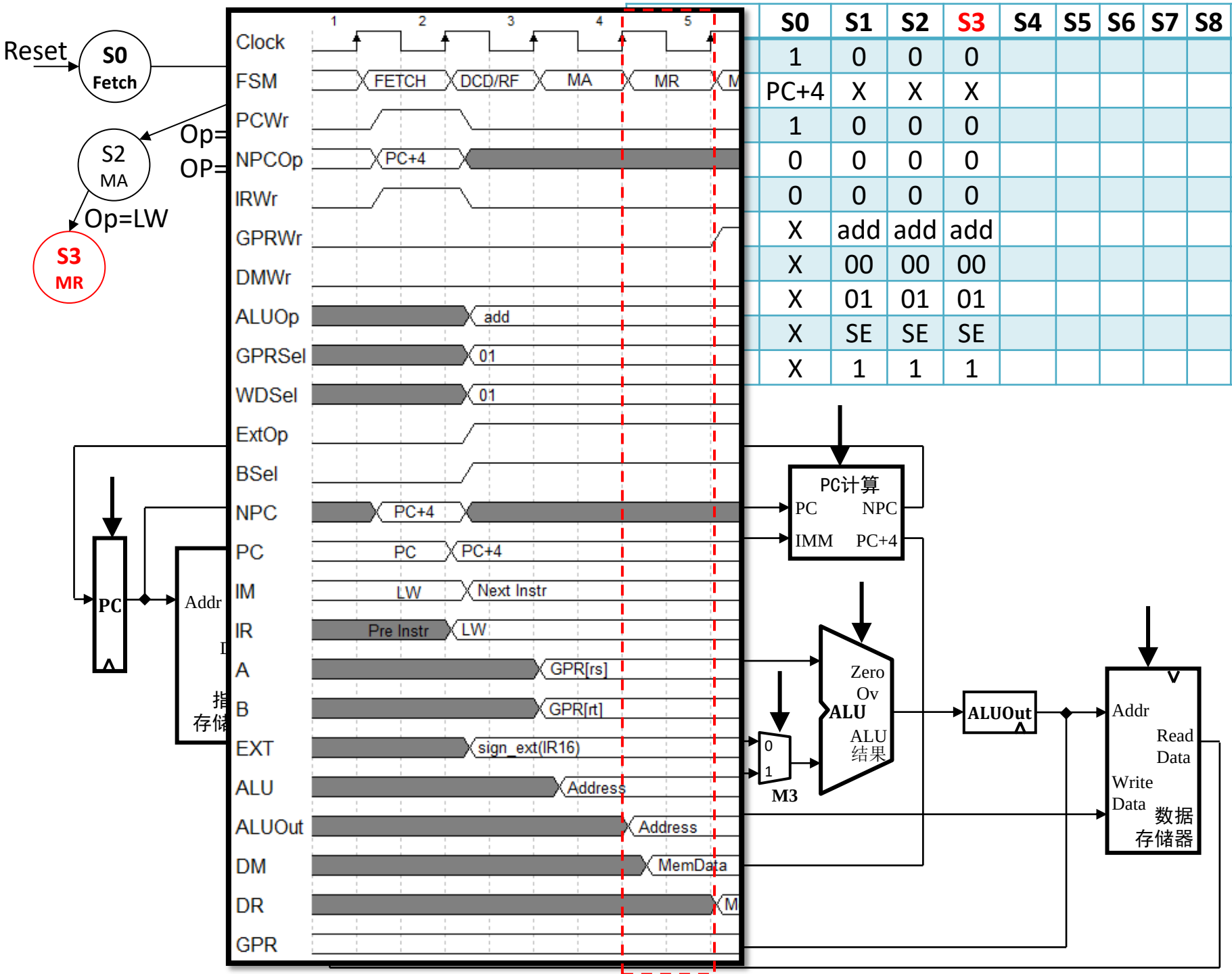
S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
1	0	0							
PC+4	X	X							
1	0	0							
0	0	0							
0	0	0							
X	add	add							
X	00	00							
X	01	01							
X	SE	SE							
X	1	1							



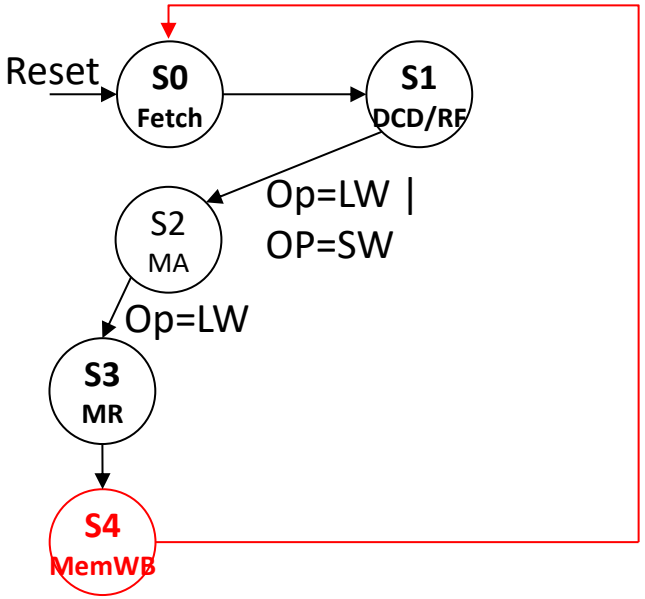


	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr	1	0	0	0						
NPCOp	PC+4	X	X	X						
IRWr	1	0	0	0						
GPRWr	0	0	0	0						
DMWr	0	0	0	0						
ALUOp	X	add	add	add						
GPRSel	X	00	00	00						
WDSel	X	01	01	01						
ExtOp	X	SE	SE	SE						
BSel	X	1	1	1						

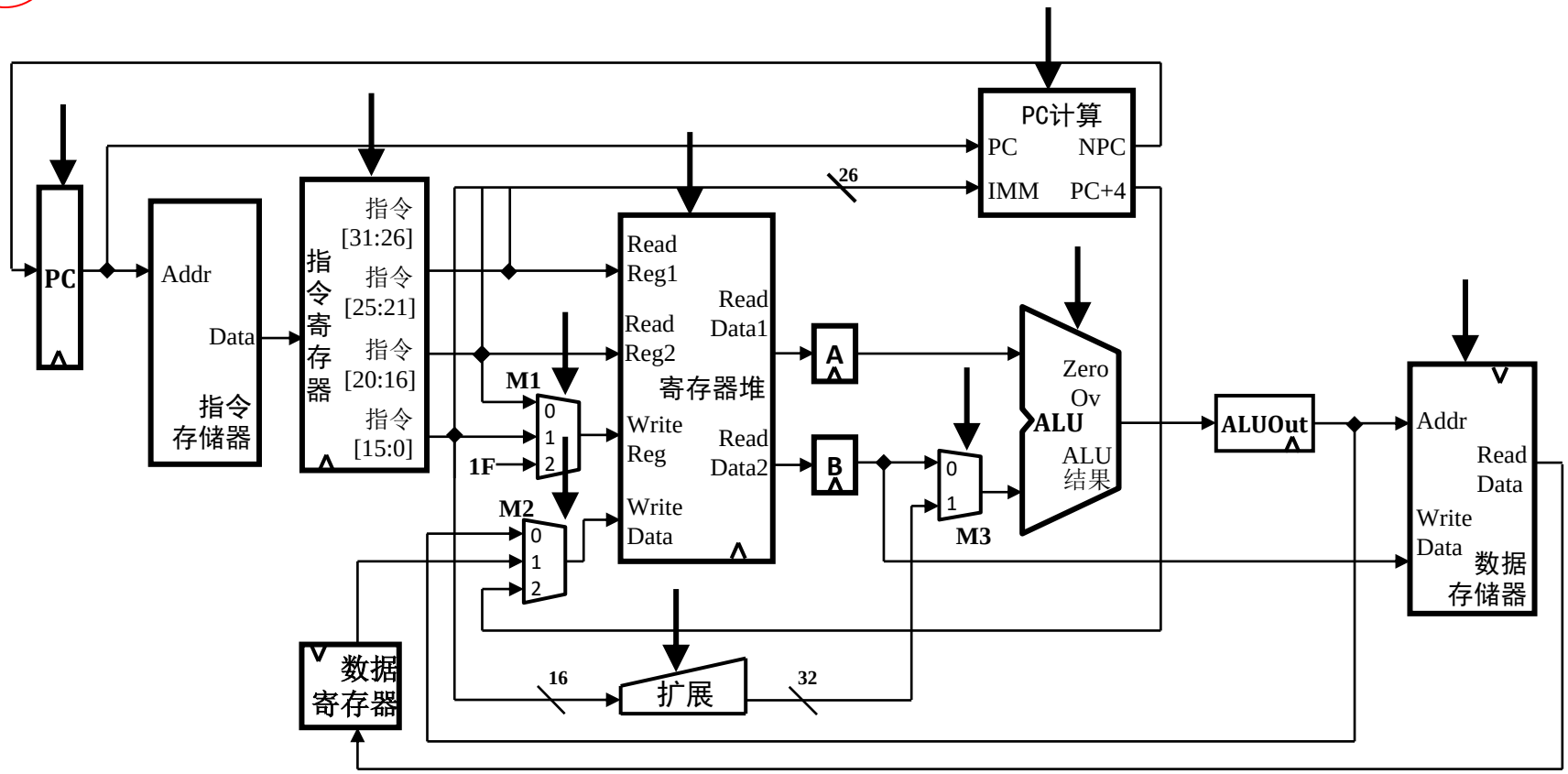


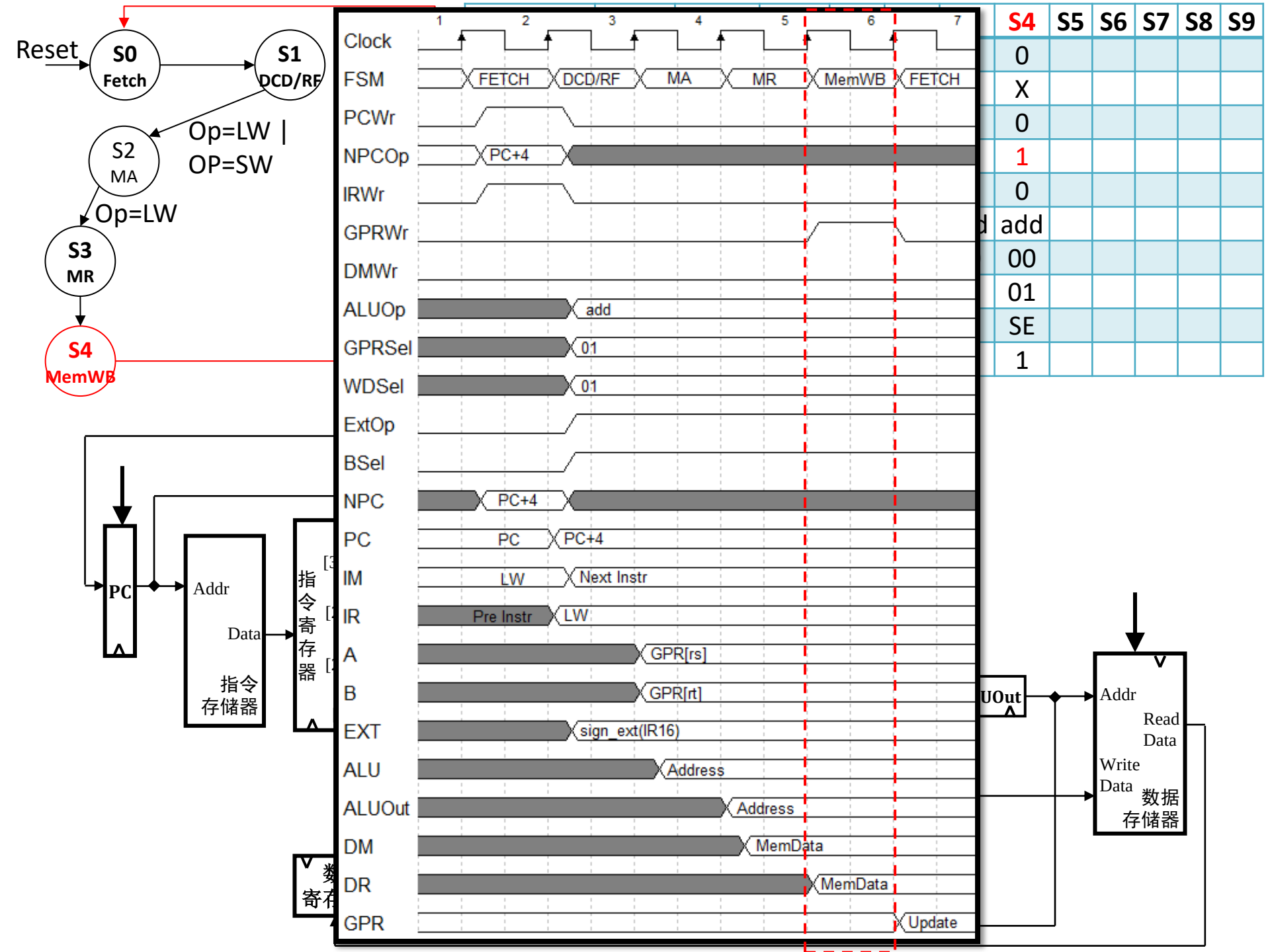


S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
1	0	0	0						
PC+4	X	X	X						
1	0	0	0						
0	0	0	0						
0	0	0	0						
X	add	add	add						
X	00	00	00						
X	01	01	01						
X	SE	SE	SE						
X	1	1	1						



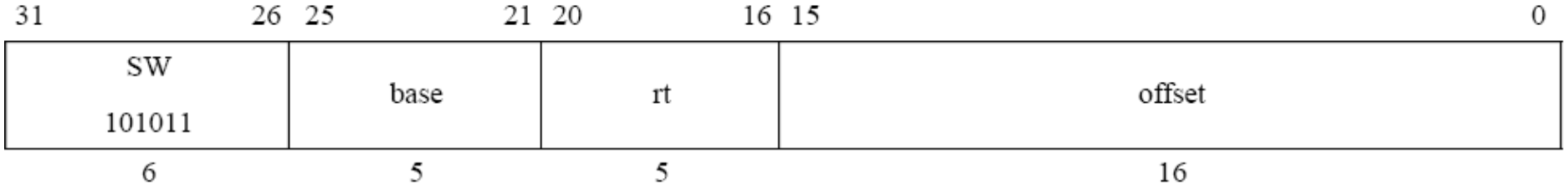
	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0	0	0	0					
NPCOp			PC+4	X	X	X	X					
IRWr			1	0	0	0	0					
GPRWr			0	0	0	0	1					
DMWr			0	0	0	0	0					
ALUOp			X	add	add	add	add					
GPRSel			X	00	00	00	00					
WDSel			X	01	01	01	01					
ExtOp			X	SE	SE	SE	SE					
BSel			X	1	1	1	1					




Reset
S0
Fetch
S1
DCD/RF
S2
MA
Op=LW | OP=SW
Op=LW
S3
MR
S4
MemWB
1
2
3
4
5
6
7
Clock
FSM
FETCH
DCD/RF
MA
MR
MemWB
FETCH
PCWr
NPCOp
PC+4
IRWr
GPRWr
DMWr
ALUOp
add
GPRSel
01
WDSel
01
ExtOp
BSel
NPC
PC+4
PC
PC
PC+4
IM
LW
Next Instr
IR
Pre Instr
LW
A
GPR[rs]
B
GPR[rt]
EXT
sign_ext(IR16)
ALU
Address
ALUOut
Address
DM
MemData
DR
MemData
GPR
Update
S4 S5 S6 S7 S8 S9
0
X
0
1
0
add
00
01
SE
1
PC
Addr
Data
指令存储器
指令寄存器
UOut
Addr
Read Data
Write Data
数据存储器

FSM构造过程

- LW, SW, ADDU, SUBU, ORI, LUI, BEQ, JAL

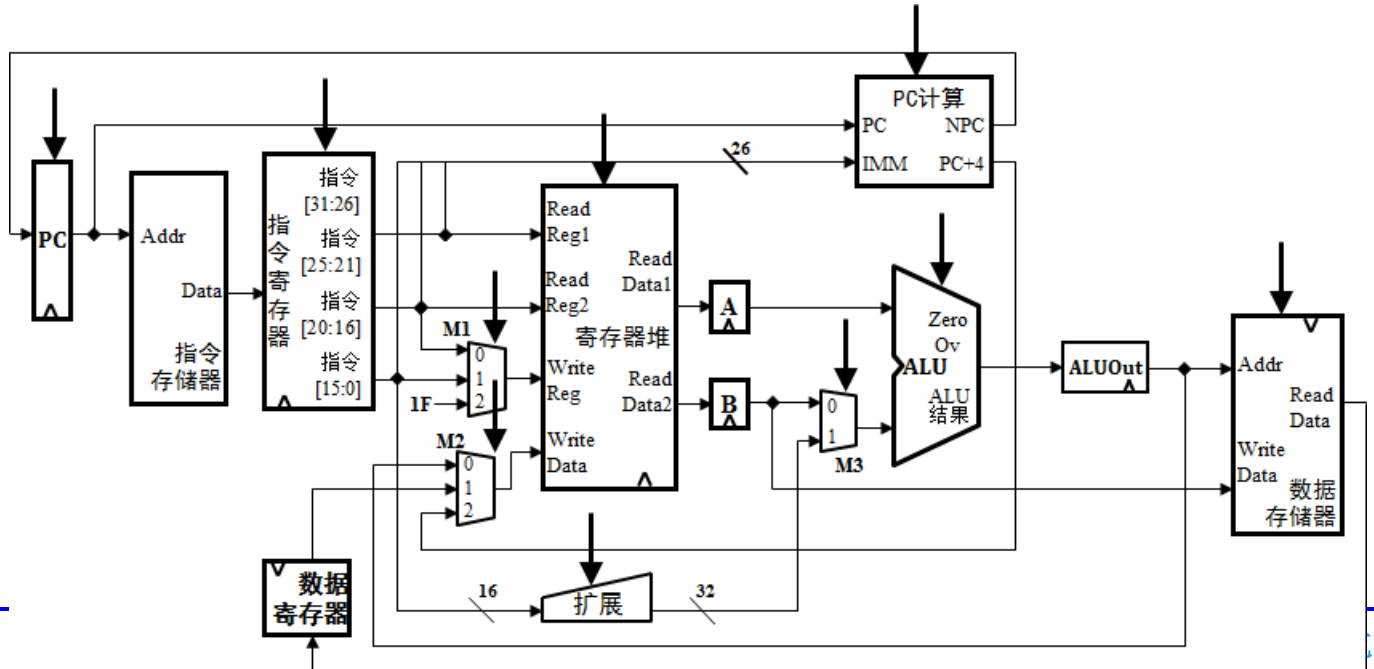


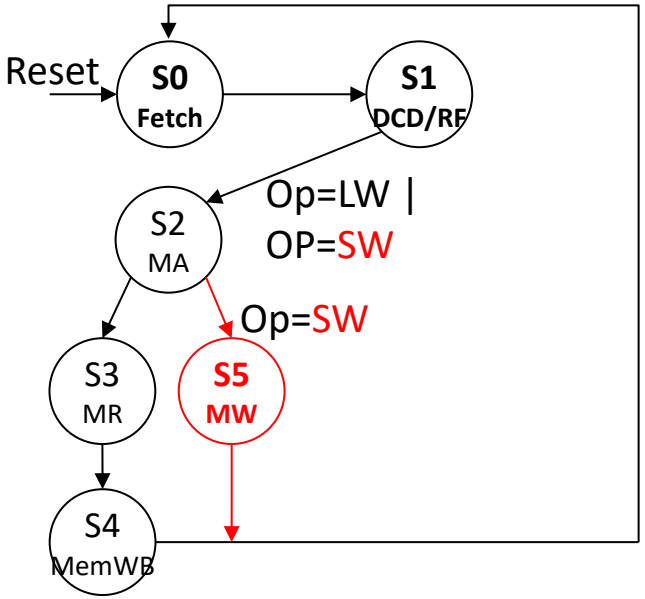

```
Addr ← sign_extend(offset) + GPR[base]
memword ← Memory[Addr]
GPR[rt] ← memword
PC ← PC + 4
```

RTL描述

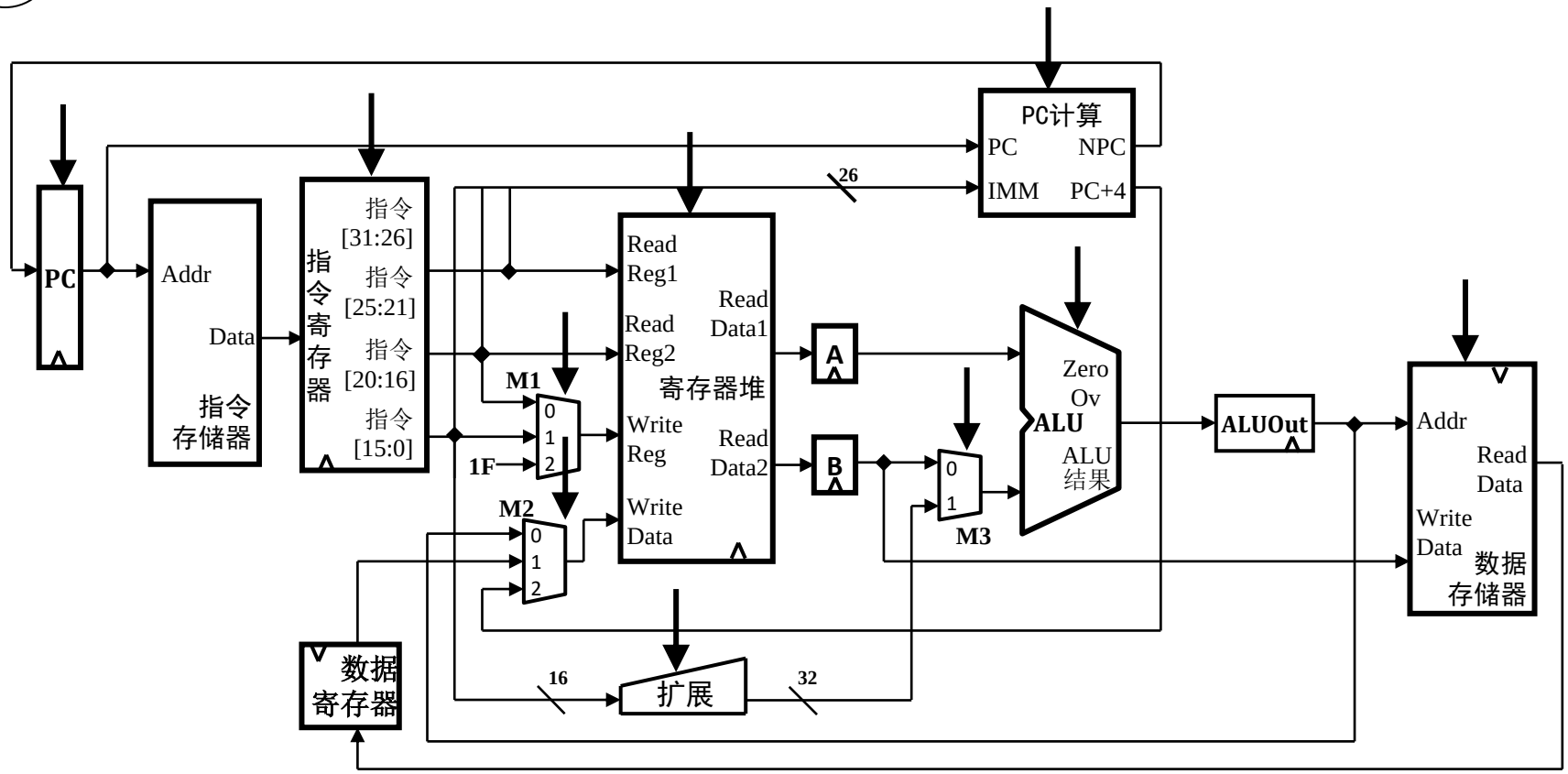
MEM[R[rs]+sign_ext(imm16)] ←R[rt]

PC←PC+4





	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0	0		0					
NPCOp			PC+4	X	X		X					
IRWr			1	0	0		0					
GPRWr			0	0	0		0					
DMWr			0	0	0		1					
ALUOp			X	add	add		add					
GPRSel			X	00	00		00					
WDSel			X	01	01		01					
ExtOp			X	SE	SE		SE					
BSel			X	1	1		1					



FSM构造过程

- LW，SW，ADDU，SUBU，ORI，LUI，BEQ，JAL

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL 000000	rs	rt	rd	0 00000	ADDU 100001
6	5	5	5	5	6

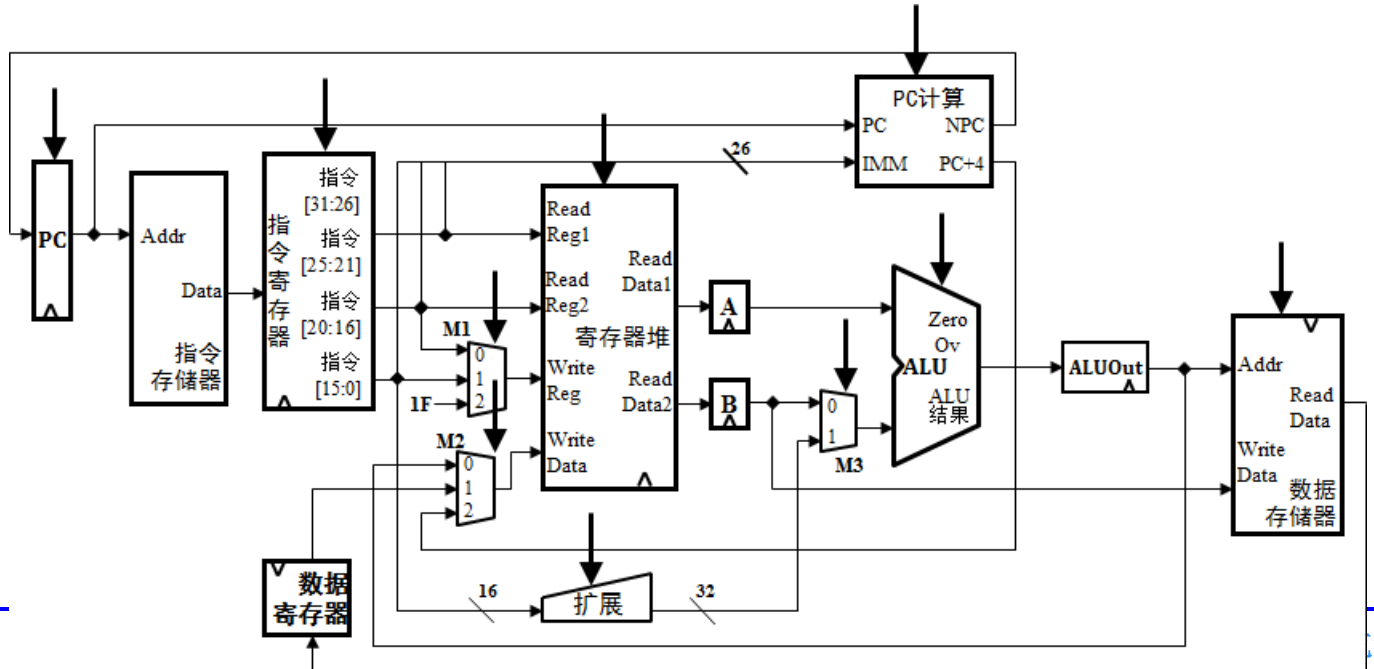
Operation:

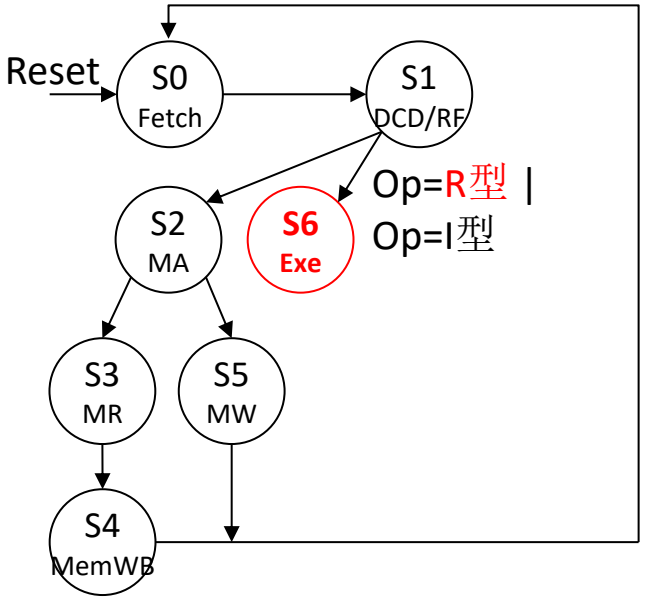
```
temp ← GPR[rs] + GPR[rt]
GPR[rd] ← temp
```

RTL

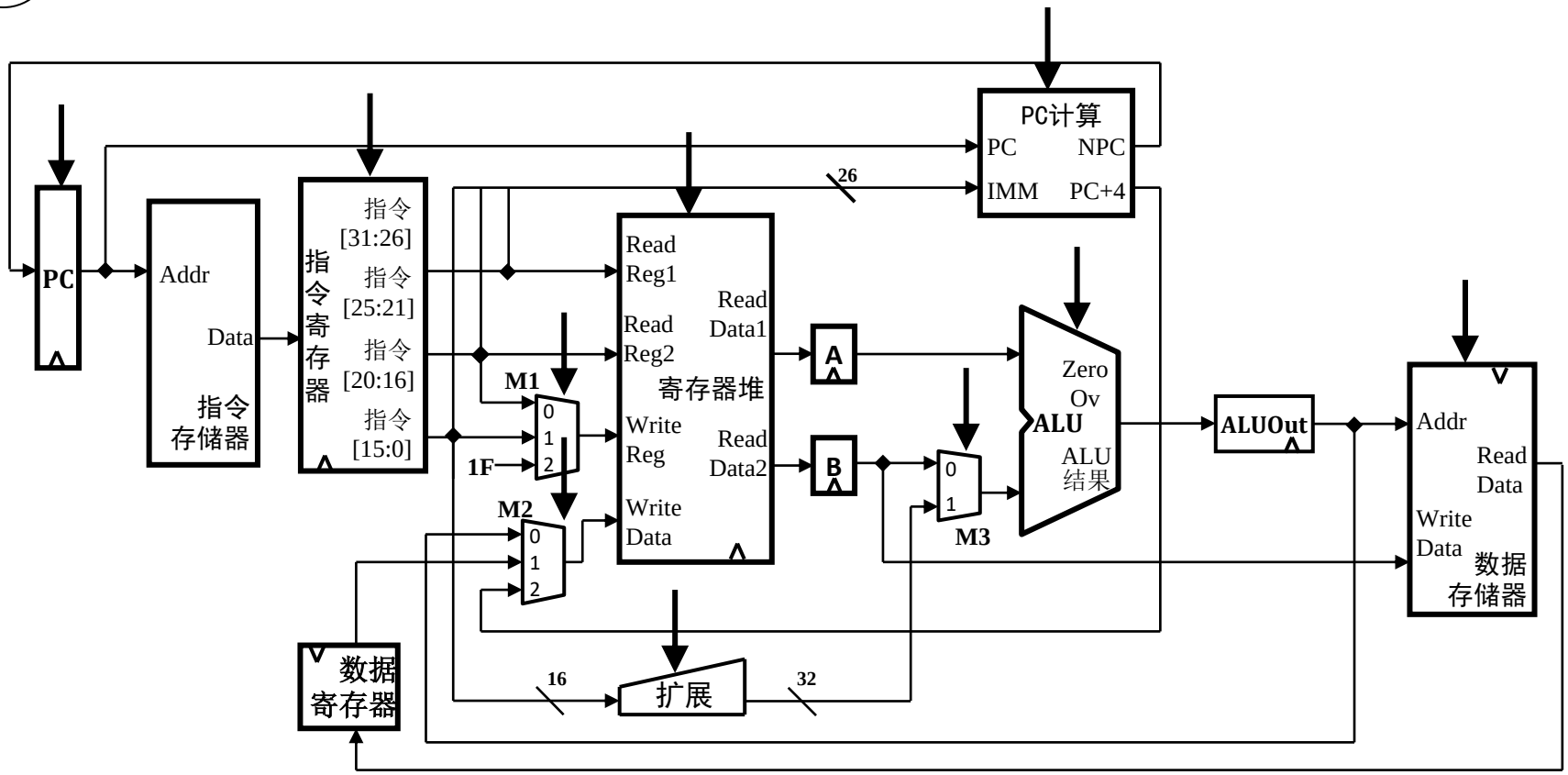
R[rd]←R[rs]+R[rt]

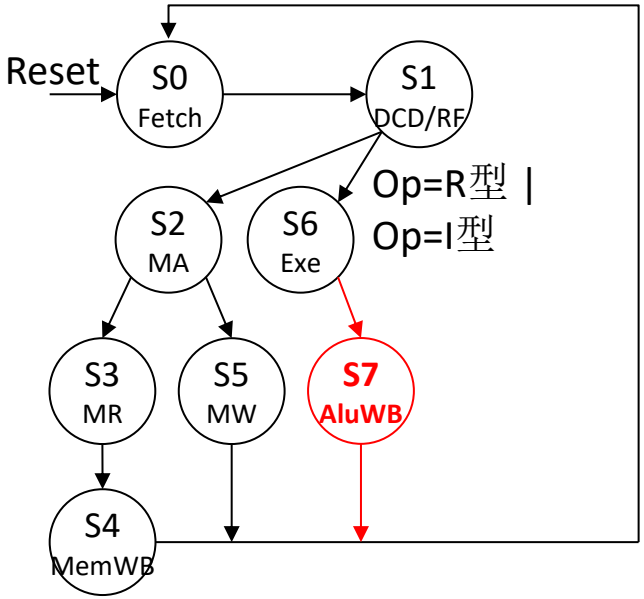
PC←PC+4



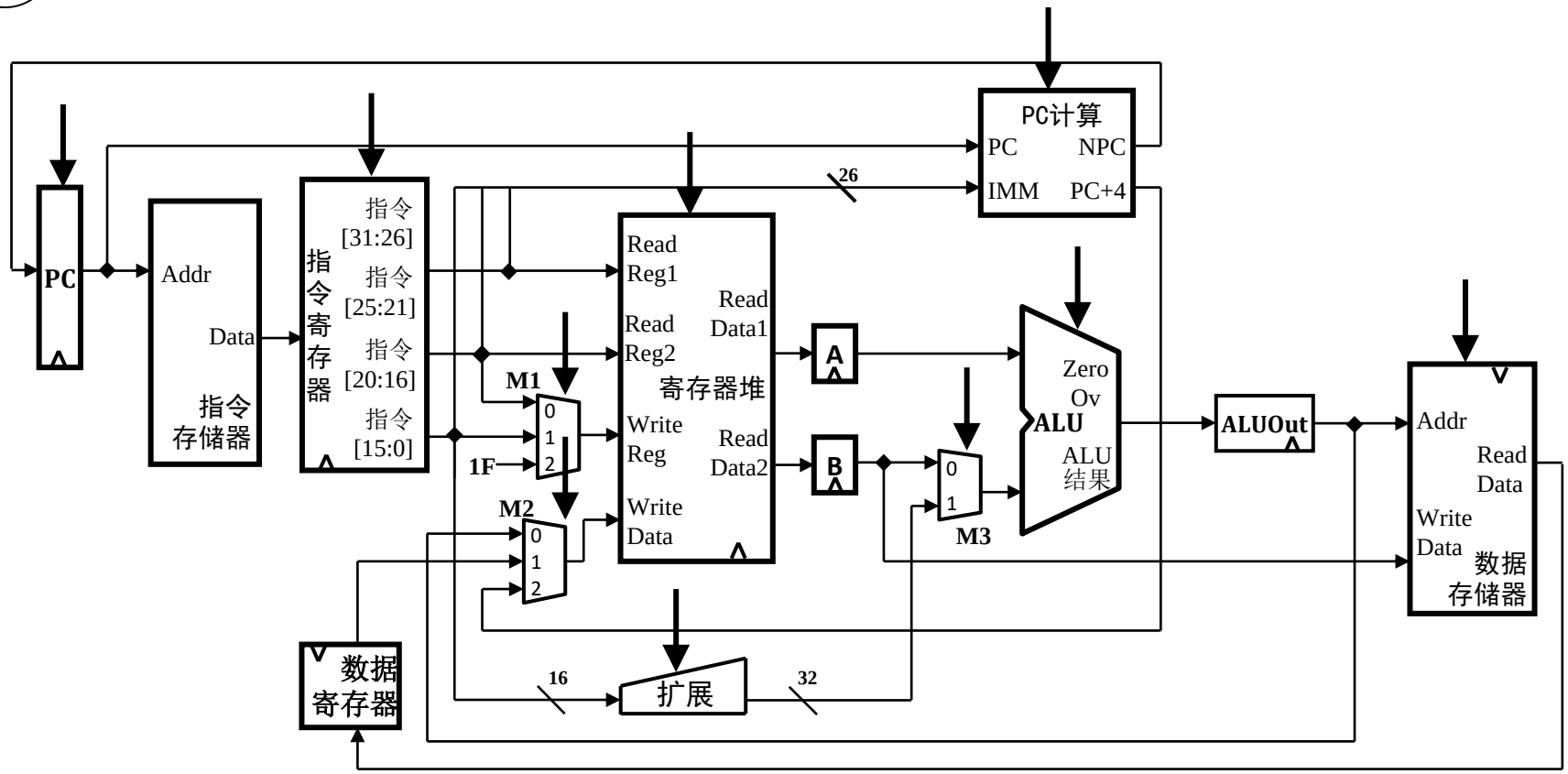


	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0					0			
NPCOp			PC+4	X					X			
IRWr			1	0					0			
GPRWr			0	0					0			
DMWr			0	0					0			
ALUOp			X	add					add			
GPRSel			X	01					01			
WDSel			X	00					00			
ExtOp			X	X					X			
BSel			X	0					0			



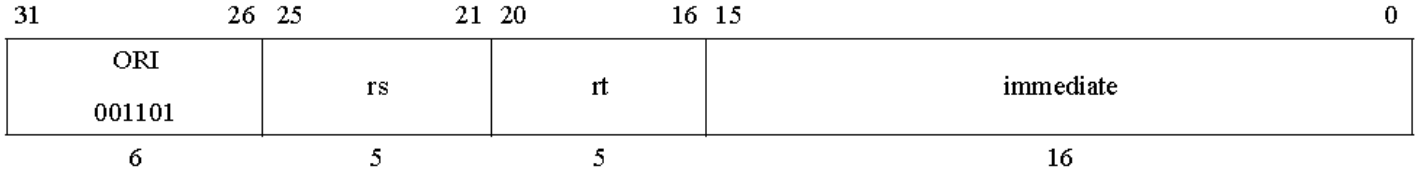


	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0					0	0		
NPCOp			PC+4	X					X	X		
IRWr			1	0					0	0		
GPRWr			0	0					0	1		
DMWr			0	0					0	0		
ALUOp			X	add					add	add		
GPRSel			X	01					01	01		
WDSel			X	00					00	00		
ExtOp			X	X					X	X		
BSel			X	0					0	0		



FSM构造过程

- LW，SW，ADDU，SUBU，ORI，LUI，BEQ，JAL



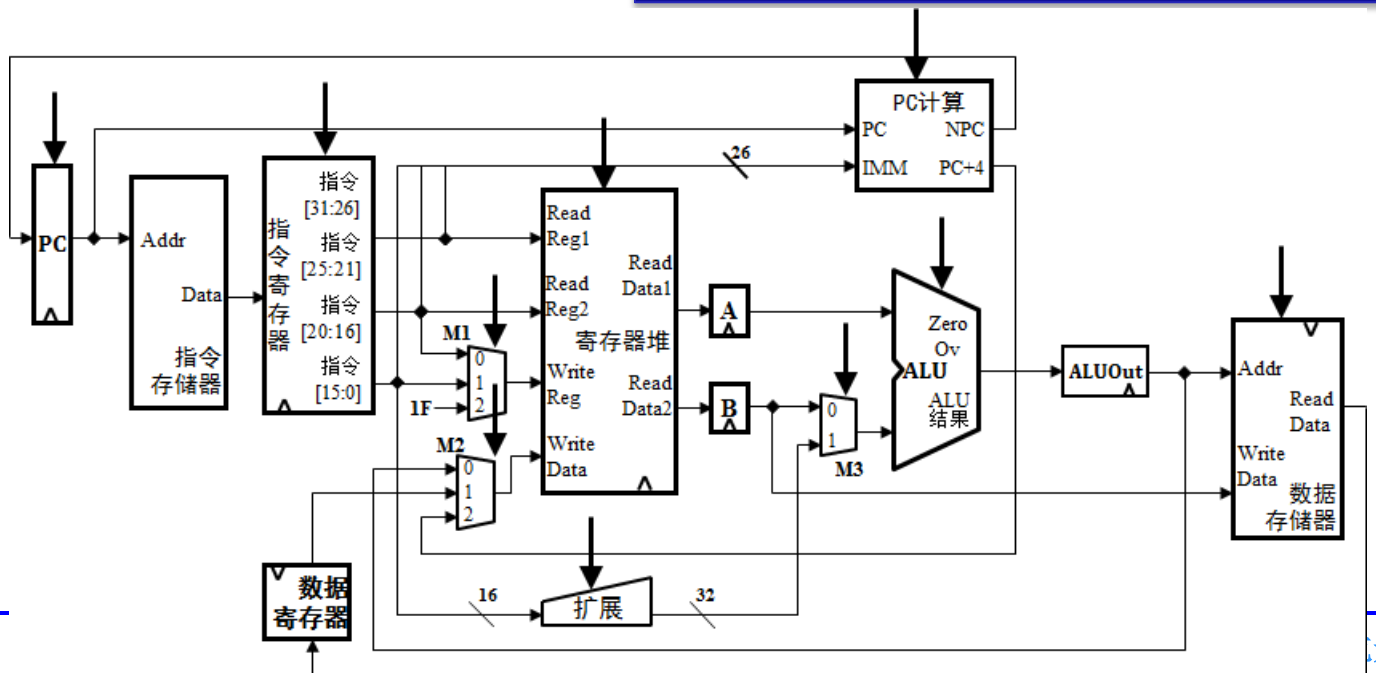
Operation:

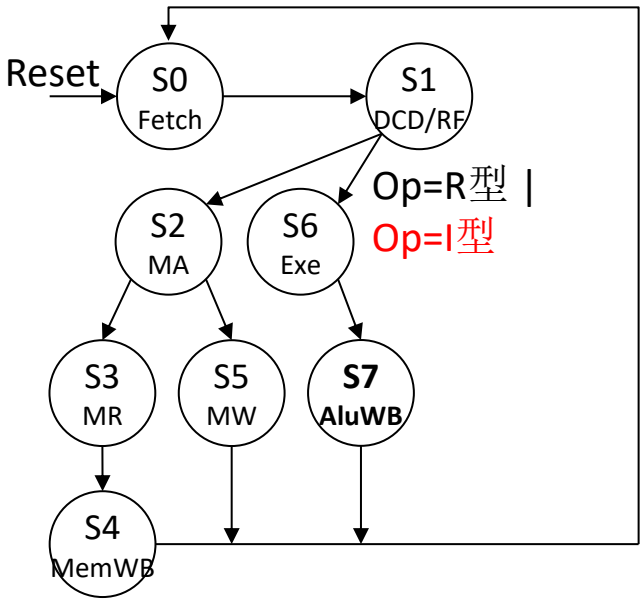
```
GPR[rt] ← GPR[rs] or zero_extend(imm
```

RTL

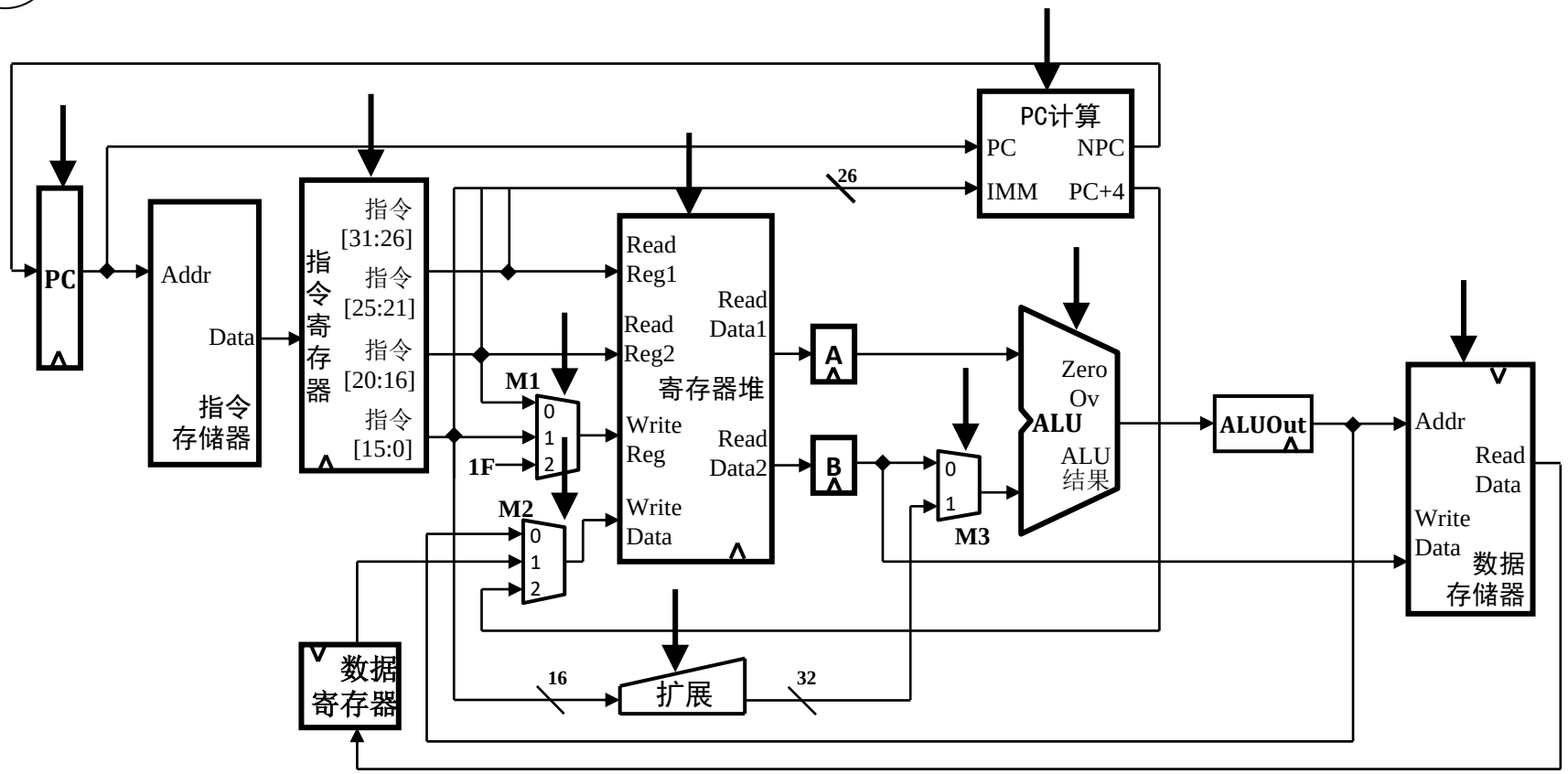
R[rd]←R[rs] | zero_extend(imm16)

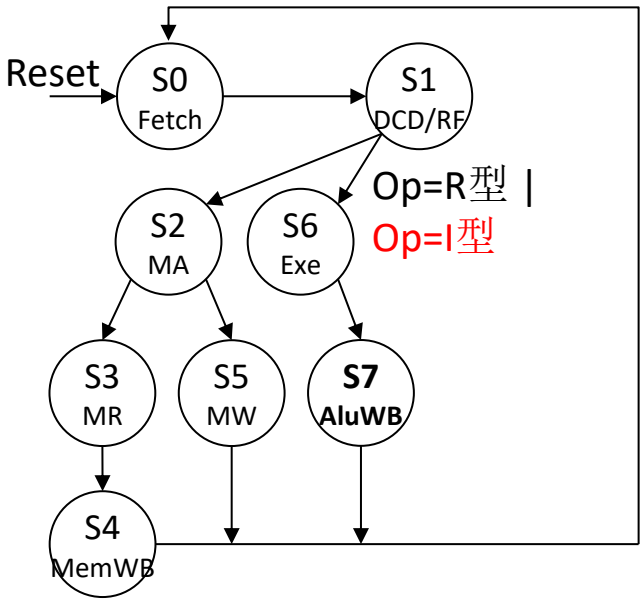
PC←PC+4





	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0					0	0		
NPCOp			PC+4	X					X	X		
IRWr			1	0					0	0		
GPRWr			0	0					0	1		
DMWr			0	0					0	0		
ALUOp			X	OR					OR	OR		
GPRSel			X	00					00	00		
WDSel			X	00					00	00		
ExtOp			X	ZE					ZE	ZE		
BSel			X	1					1	1		

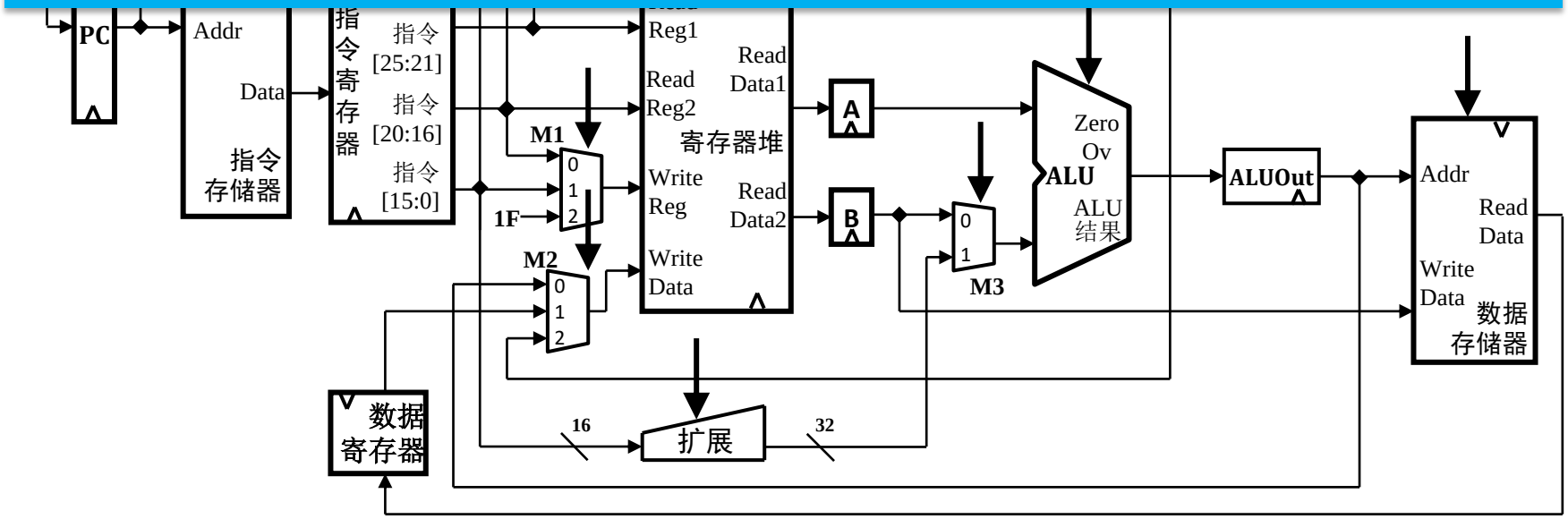




	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0					0	0		
NPCOp			PC+4	X					X	X		
IRWr			1	0					0	0		
GPRWr			0	0					0	1		
DMWr			0	0					0	0		
ALUOp			X	OR					OR	OR		
GPRSel			X	00					00	00		
WDSel			X	00					00	00		
ExtOp			X	ZE					ZE	ZE		
BSel			X	1					1	1		

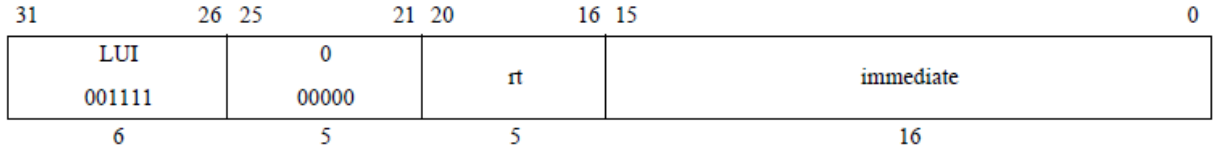
I型与R型没有实质差别！区别仅在于：

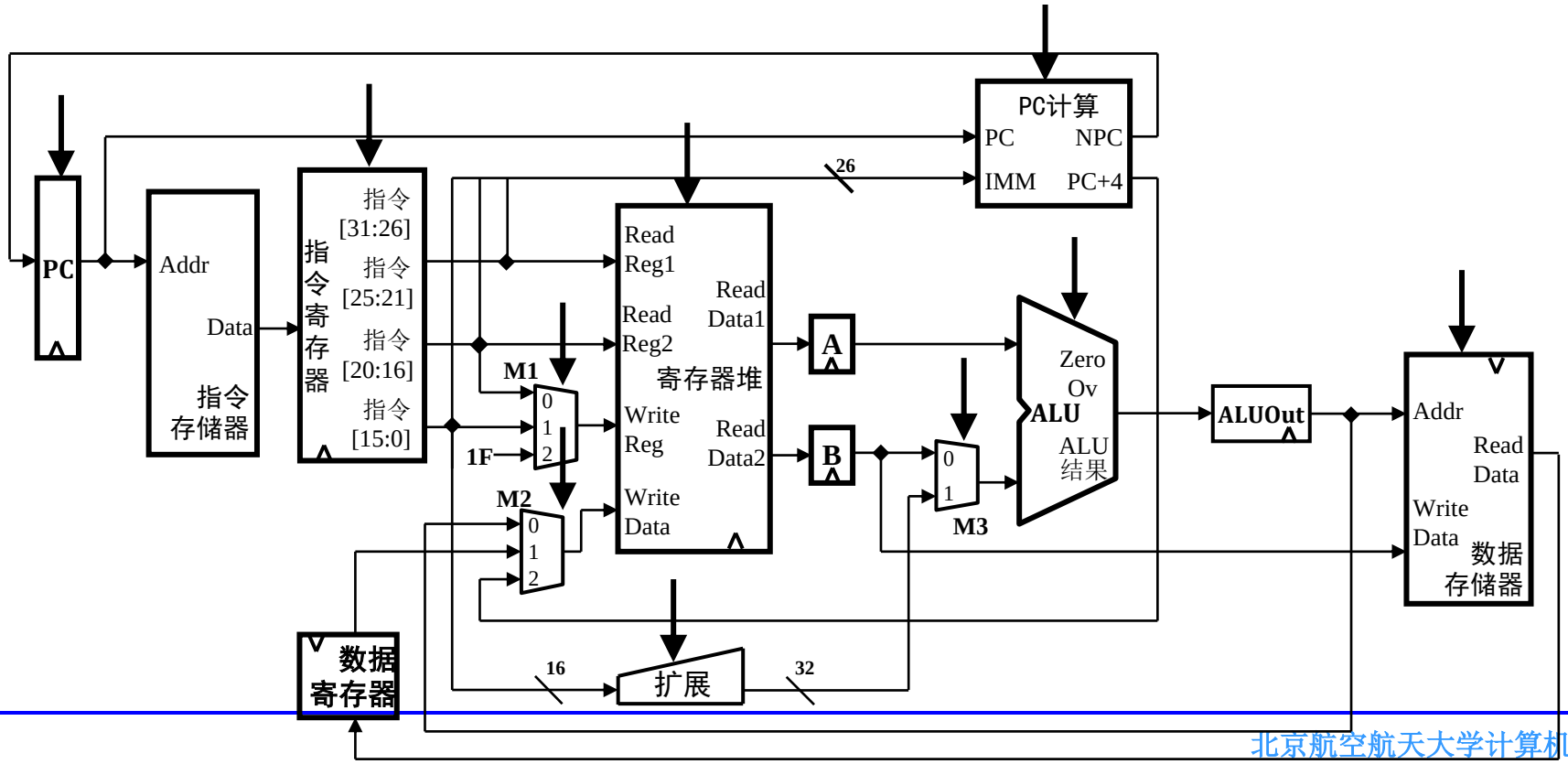
- ALU的B通道的控制

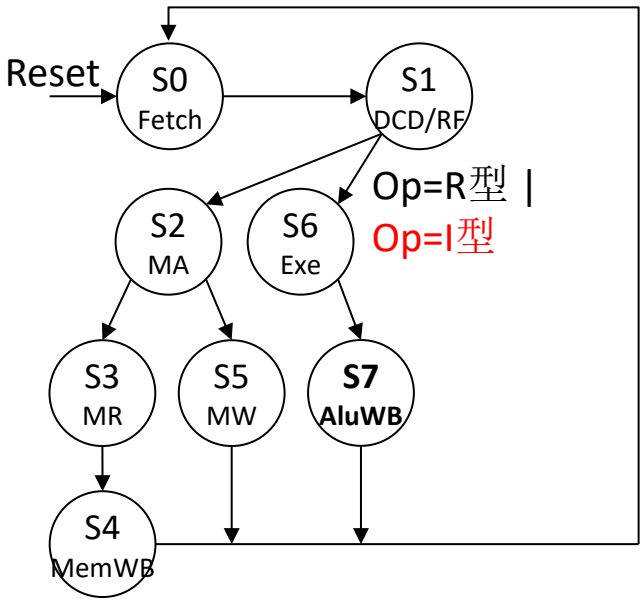


FSM构造过程

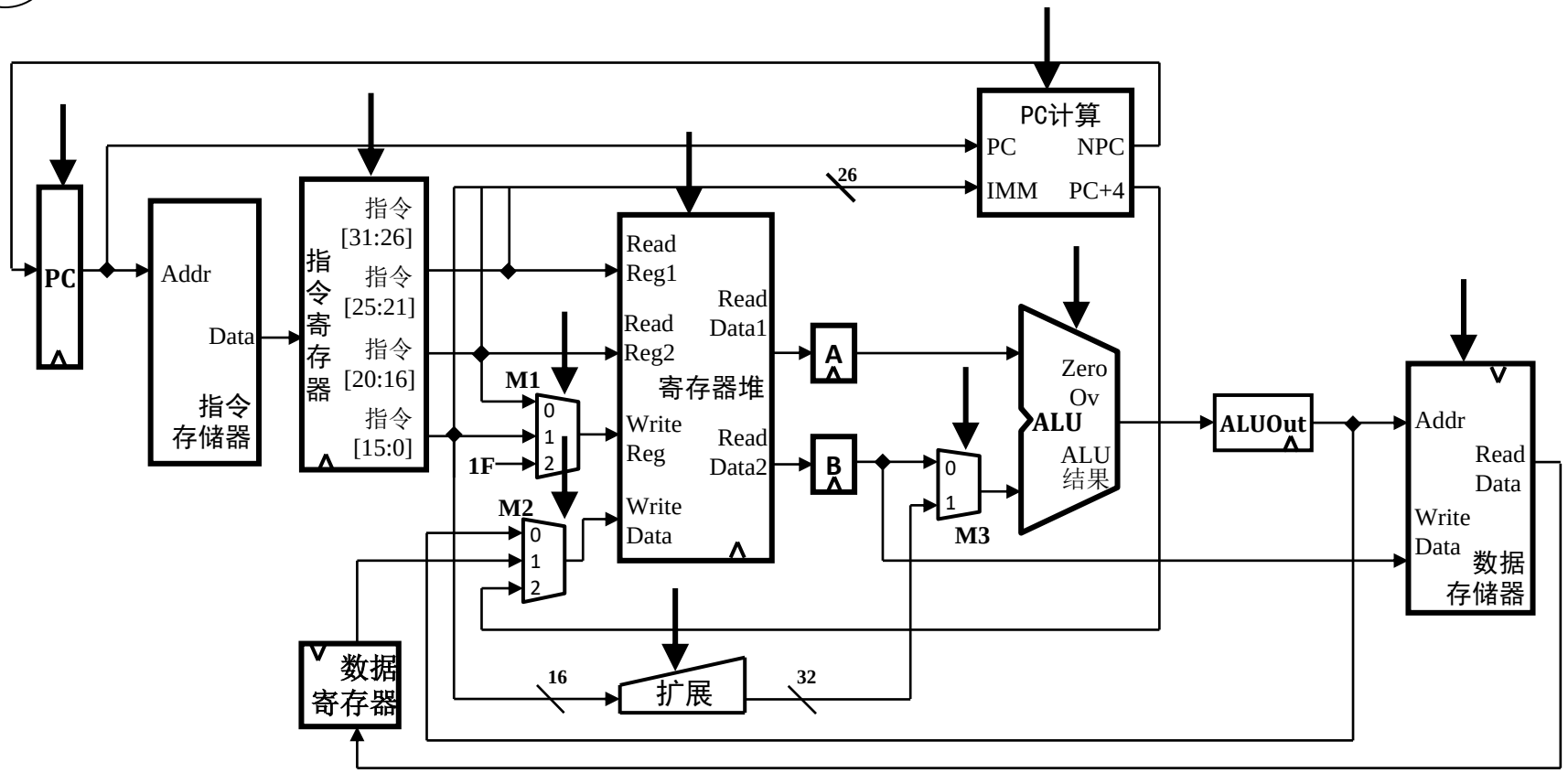
- LW，SW，ADDU，SUBU，ORI，LUI，BEQ，JAL
- LUI怎么执行？
  - GPR[rt] ← imm16 || 0^{16}

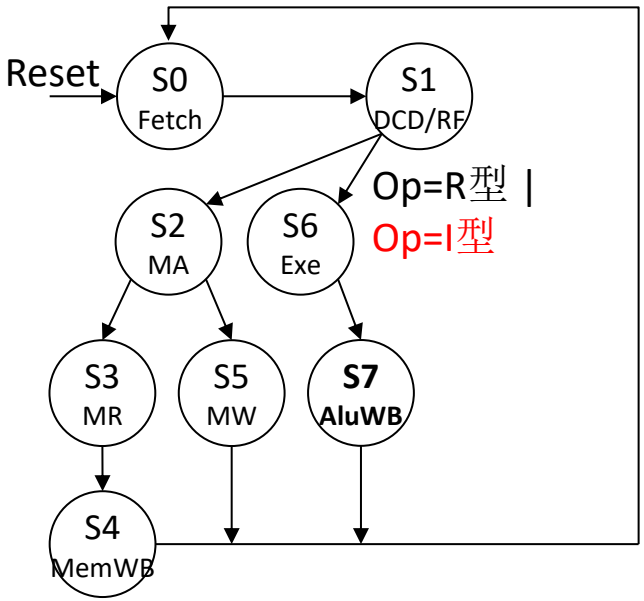






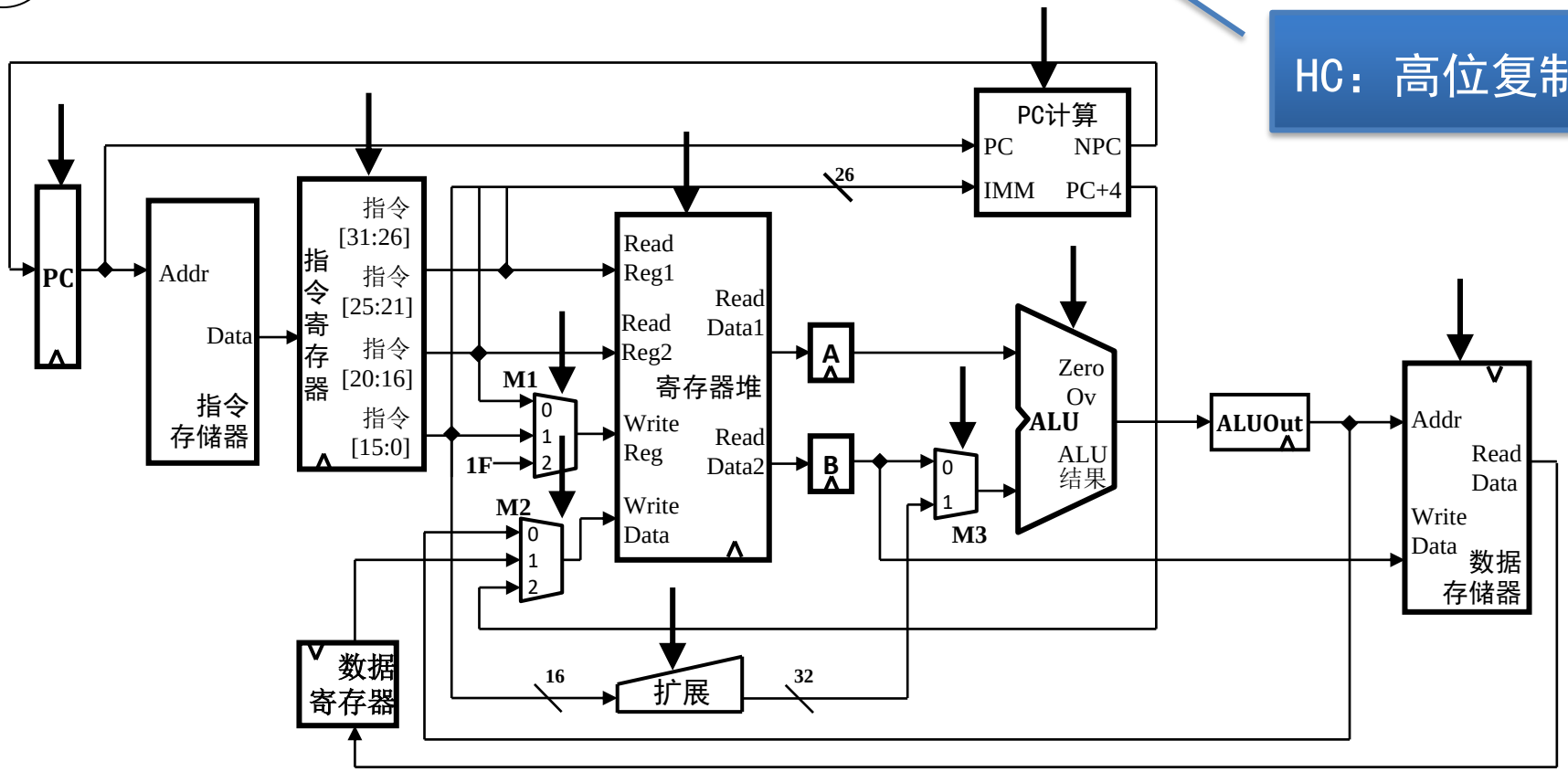
	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0								
NPCOp			PC+4	X								
IRWr			1	0								
GPRWr			0	0								
DMWr			0	0								
ALUOp			X									
GPRSel			X	00								
WDSel			X	00								
ExtOp			X									
BSel			X	1								





	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0					0	0		
NPCOp			PC+4	X					X	X		
IRWr			1	0					0	0		
GPRWr			0	0					0	1		
DMWr			0	0					0	0		
ALUOp			X	OR					OR	OR		
GPRSel			X	00					00	00		
WDSel			X	00					00	00		
ExtOp			X	HC					HC	HC		
BSel			X	1					1	1		

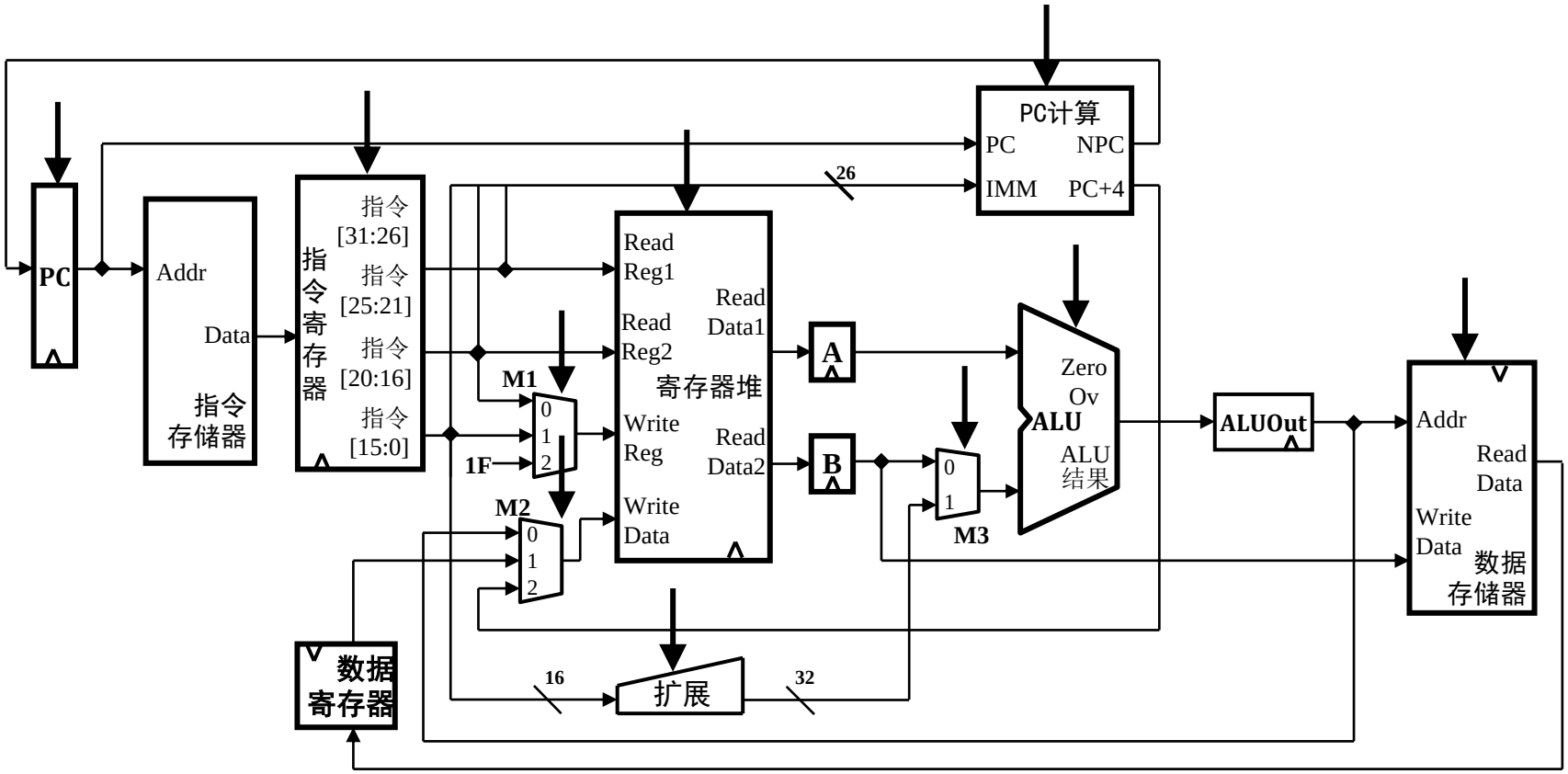
HC：高位复制

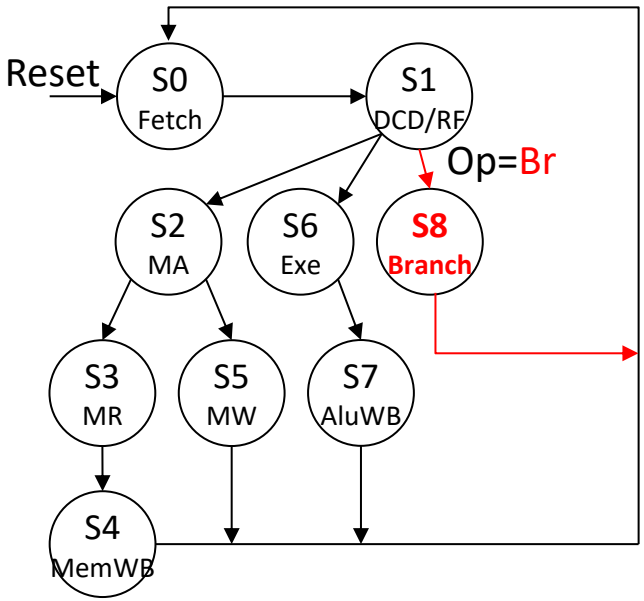


FSM构造过程

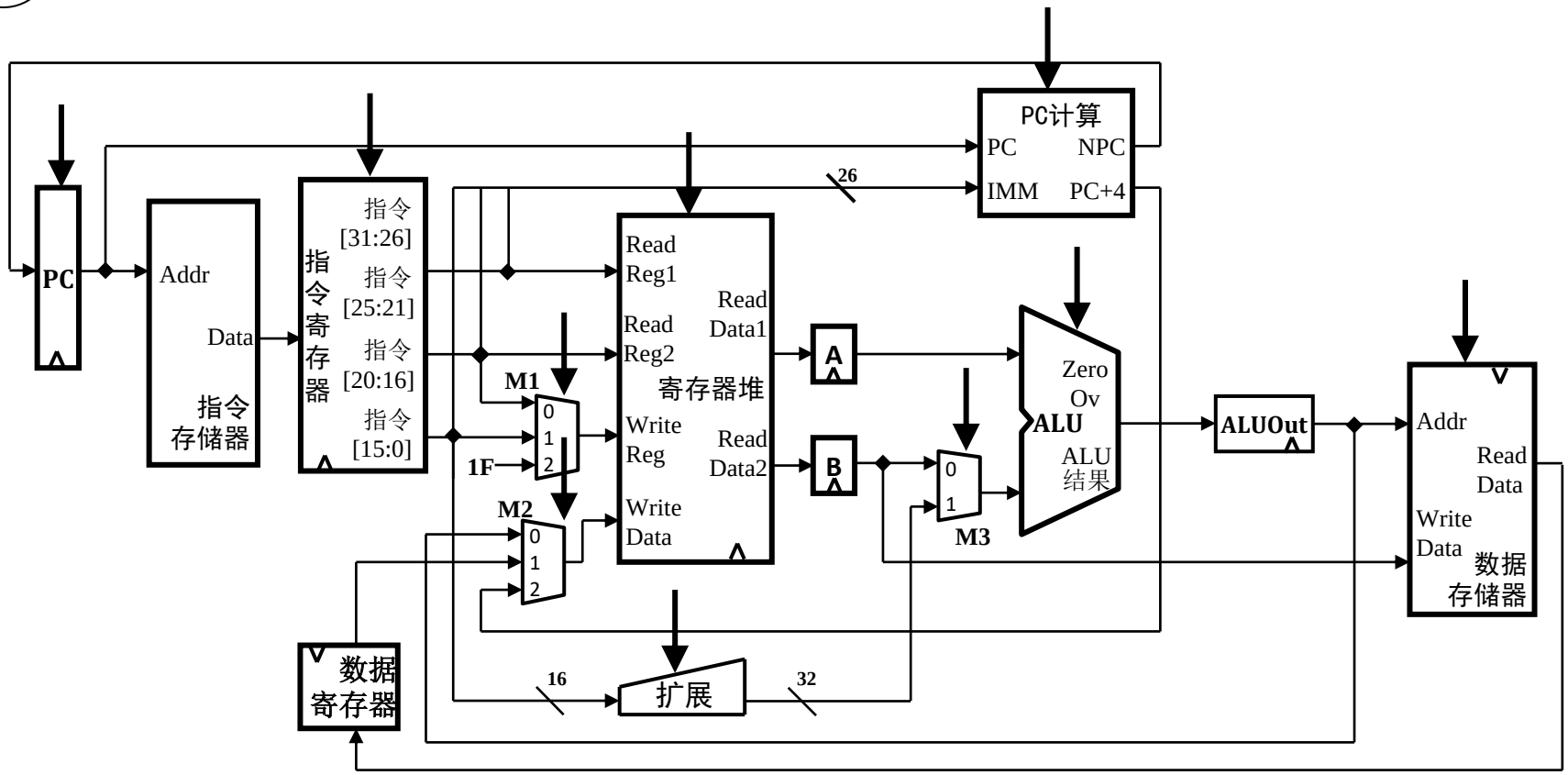
- LW，SW，ADDU，SUBU，ORI，LUI，BEQ，JAL

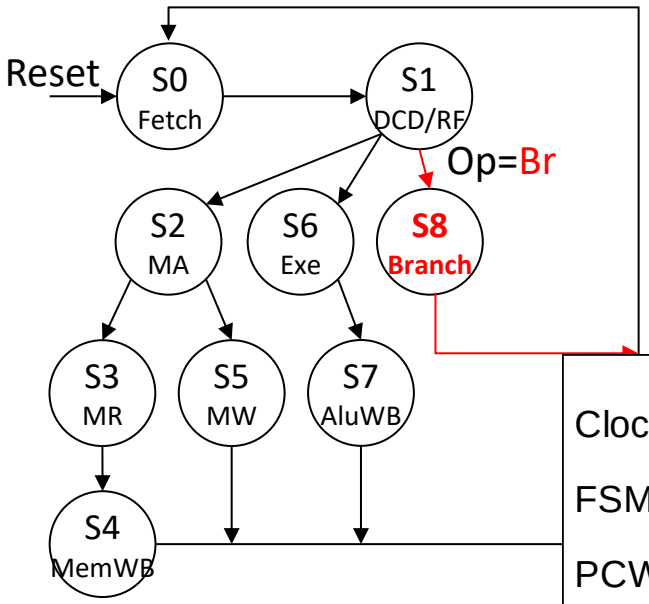
```
beq     if(R[rs]==R[rt])
           then PC←PC+4+[sign_ext(imm16)||00]
           else PC←PC+4
```



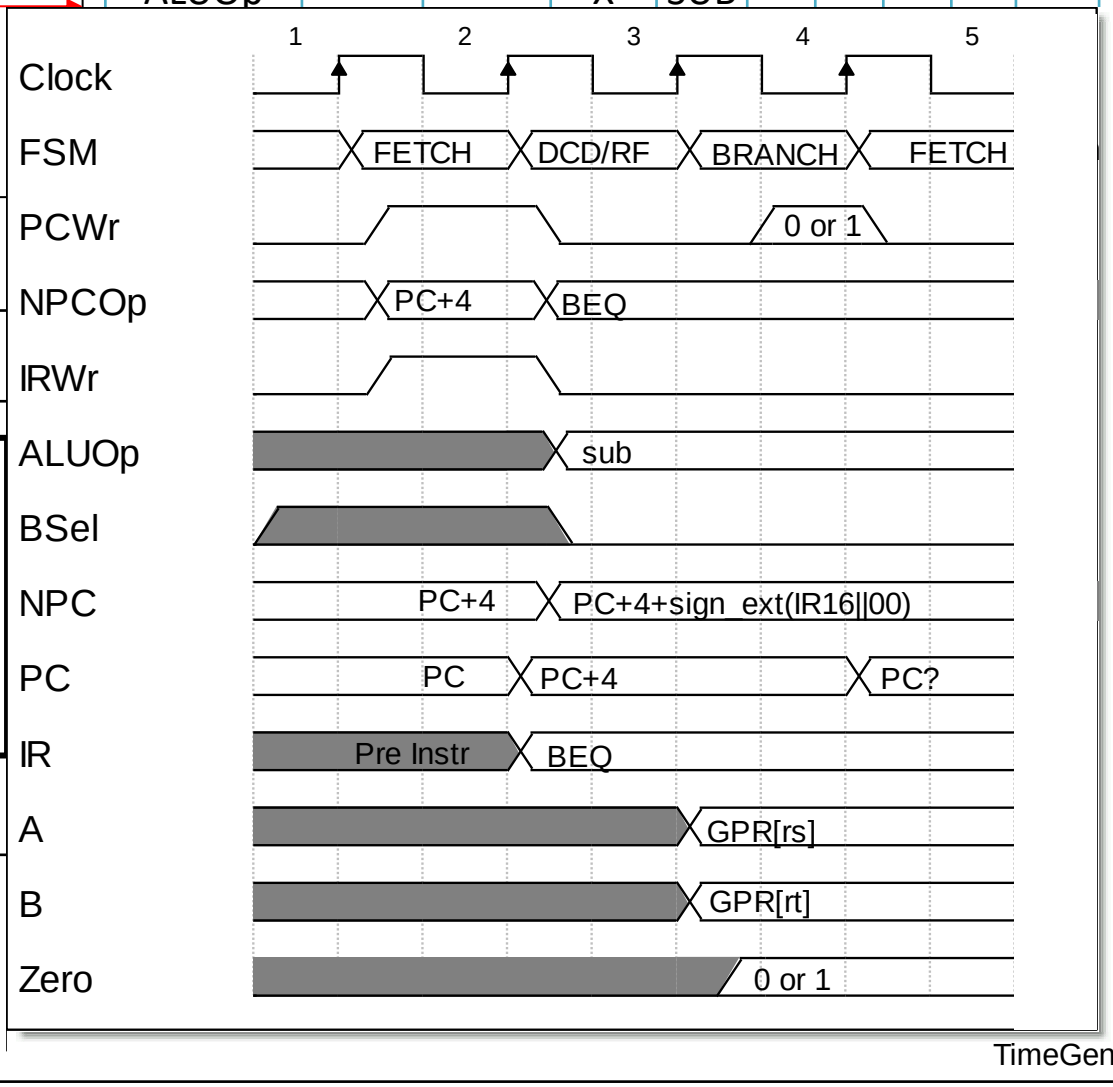


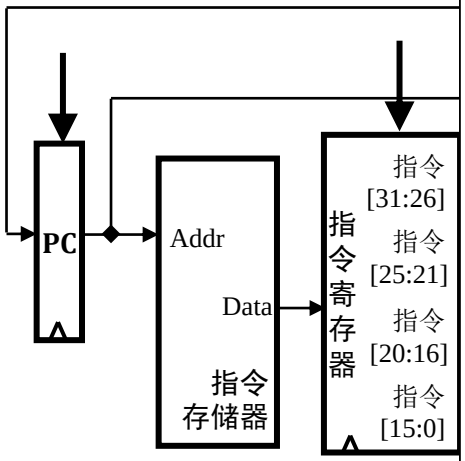
	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0							0/1	
NPCOp			PC+4	BEQ								
IRWr			1	0								
GPRWr			0	0								
DMWr			0	0								
ALUOp			X	SUB							SUB	
GPRSel			X	X							X	
WDSel			X	X							X	
ExtOp			X	X							X	
BSel			X	0							0	

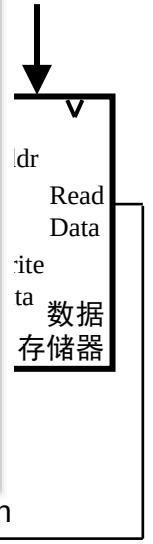




	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0							0/1	
NPCOp			PC+4	BEQ								
IRWr			1	0								
GPRWr			0	0								
DMWr			0	0								
ALUOp			X	SUB							SUB	
											X	
											X	
											X	
											0	





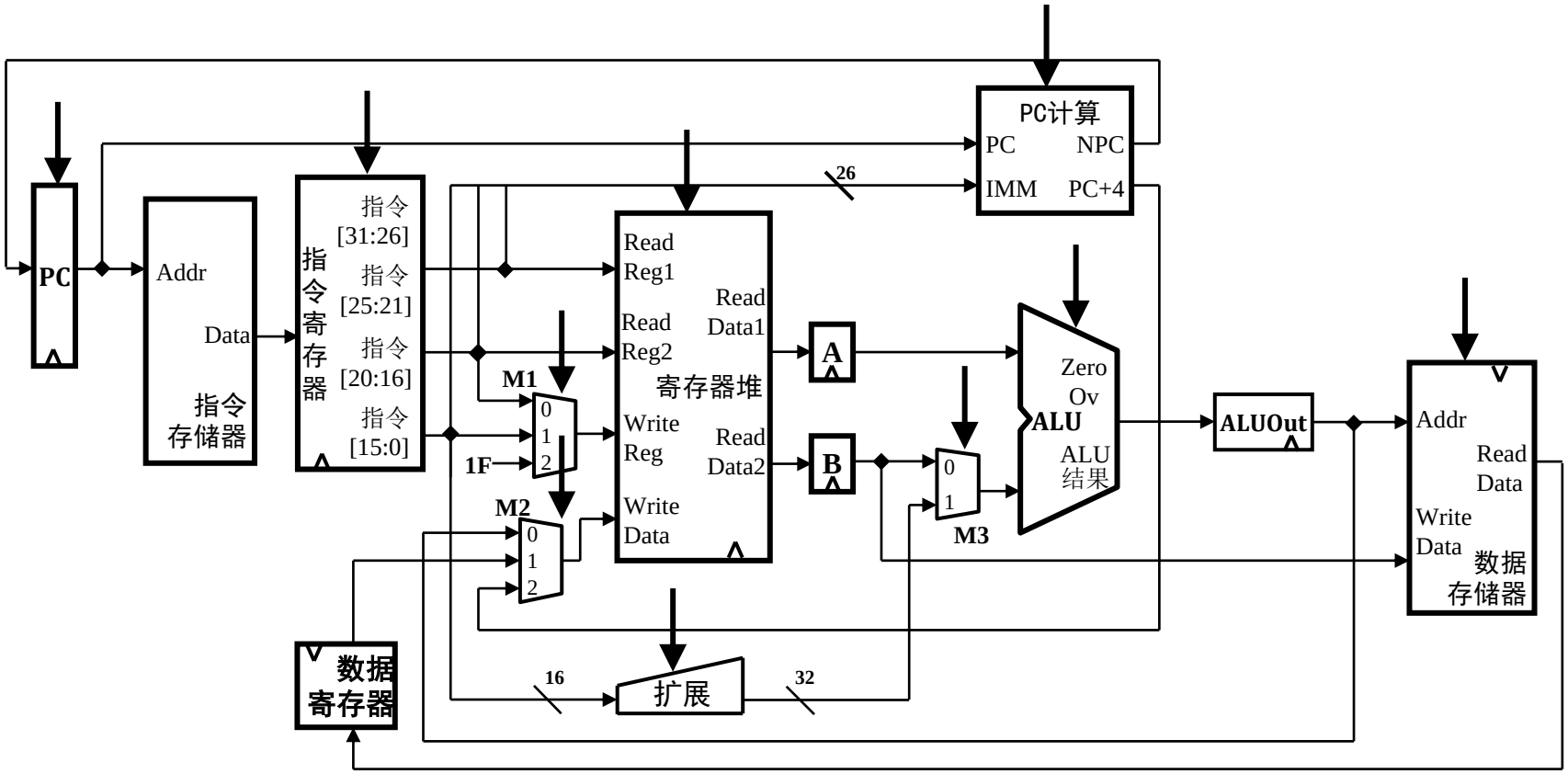


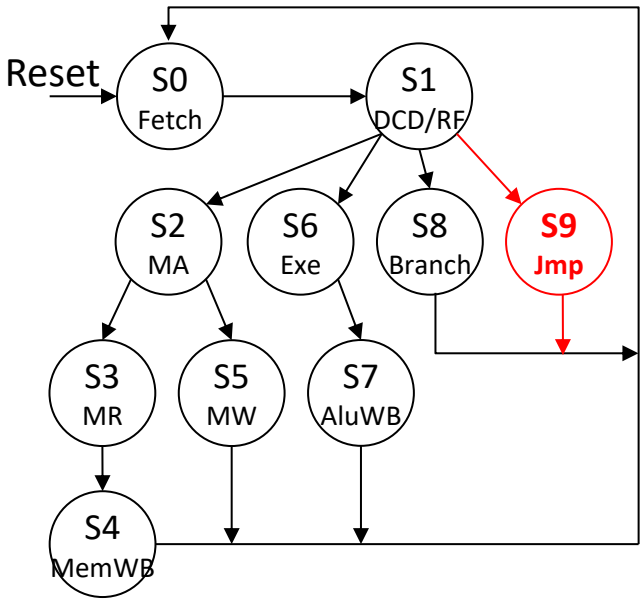
TimeGen

FSM构造过程

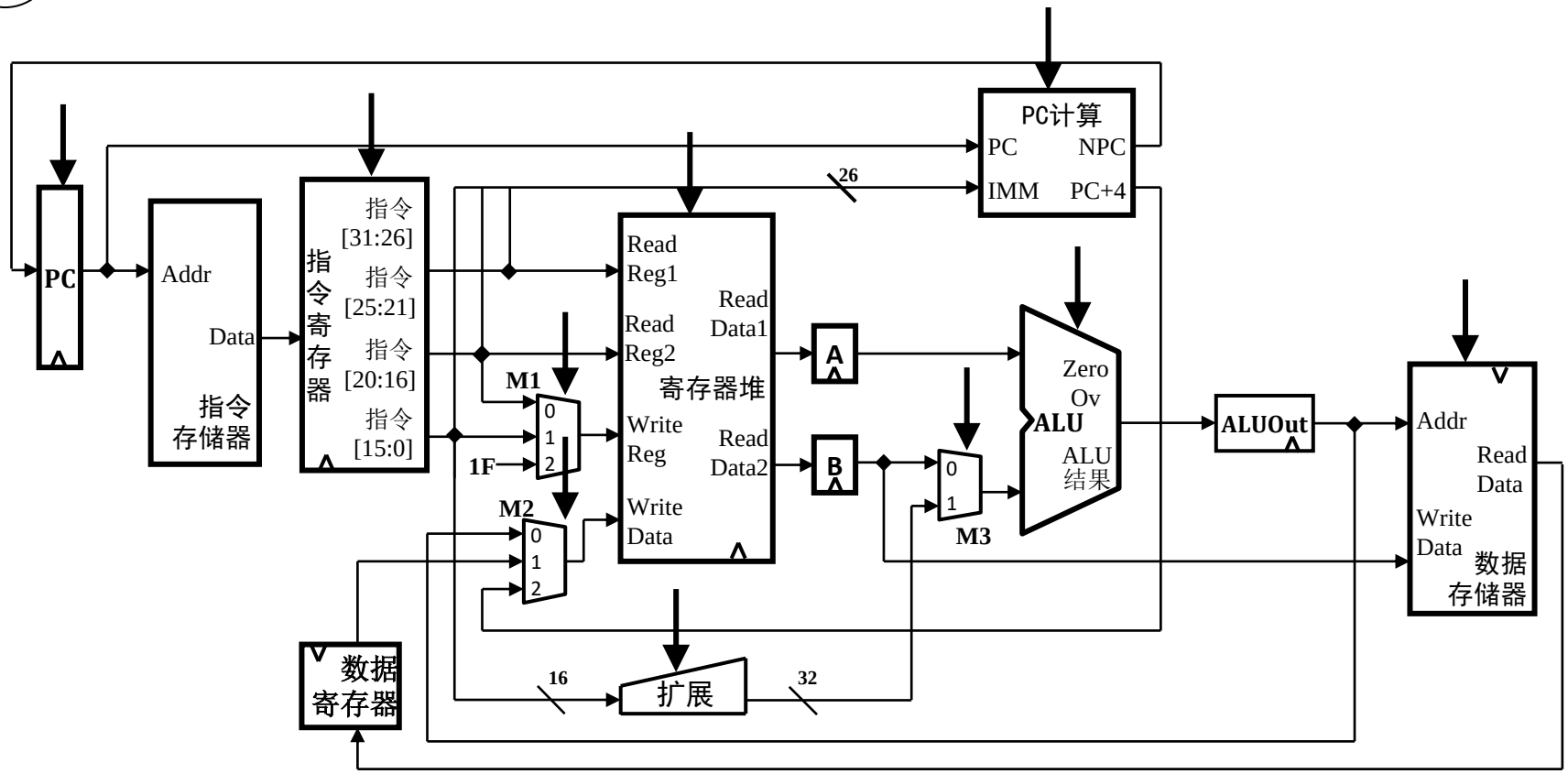
- LW，SW，ADDU，SUBU，ORI，LUI，BEQ，JAL

```
I:   GPR[31] ← PC + 8
I+1: PC      ← PC_GPRLEN-1..28 || instr_index || 0^2
```





	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr			1	0								1
NPCOp			PC+4	BEQ								JAL
IRWr			1	0								0
GPRWr			0	0								1
DMWr			0	0								0
ALUOp			X	X								X
GPRSel			X	10								10
WDSel			X	10								10
ExtOp			X	X								X
BSel			X	X								X



综合信号：第1步

- 用变量表达所有指令

```
beq = op[5]'·op[4]'·op[3]'·op[2]·op[1]'·op[0]'

。。。

Rtype = op[5]'·op[4]'·op[3]'·op[2]'·op[1]'·op[0]'

addu = Rtype·funct[5] ·funct[4]'·funct[3]'·
                funct[2]'·funct[1]'·funct[0]
```

	Op	Funct
LW	100011	
SW	101011	
ADDU	000000	100001
SUBU	000000	100011
ORI	001101	
LUI	001111	
BEQ	000100	
JAL	000011	

综合信号：第2步

- 给状态机分配寄存器
  - S0～S9：4个寄存器，fsm[3:0]
- 定义状态编号
- 用变量表达状态

```
s0 = fsm[3]'·fsm[2]'·fsm[1]'·fsm[0]'
s1 = fsm[3]'·fsm[2]'·fsm[1]'·fsm[0]
s2 = fsm[3]'·fsm[2]'·fsm[1] ·fsm[0]'
…
s8 = fsm[3] ·fsm[2]'·fsm[1]'·fsm[0]'
s9 = fsm[3] ·fsm[2]'·fsm[1]'·fsm[0]
```

状态名	编号
S0	0000
S1	0001
S2	0010
S3	0011
S4	0100
S5	0101
S6	0110
S7	0111
S8	1000
S9	1001

综合信号：第3步(以PCWr为例)

	Op	Funct	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
LW	100011		1	0	0	0	0	0	0	0	0	0
SW	101011		1	0	0	0	0	0	0	0	0	0
ADDU	000000	100001	1	0	0	0	0	0	0	0	0	0
SUBU	000000	100011	1	0	0	0	0	0	0	0	0	0
ORI	001101		1	0	0	0	0	0	0	0	0	0
LUI	001111		1	0	0	0	0	0	0	0	0	0
BEQ	000100		1	0	0	0	0	0	0	0	0/1	0
JAL	000011		1	0	0	0	0	0	0	0	0	1

```
PCWr = (lw+sw+addu+subu+ori+lui+beq+jal)·S0 +
       beq·zero +
       jal·s9
```

合并

- 构造出N类表
  - N << 56!
- 示例：PCWr
  - 抽取：N张表分别抽取PCWr的所有真值
  - 真值：形成PCWr的真值表
  - 方程：真值表化简

注意：

⇨ 某些信号(ALUOp)可能需要二次真值表

ADD

	Op	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr		1	0					0	0		

SW

	Op	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr		1	0	0			0				

LW

	Op	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9
PCWr		1	0	0	0	0					
IRWr		1	0	0	0	0					
RFDataSrc	01										
RegDst	00										
ExtOp	S										
RegWr		0	0	0	0	1					
ALUSrcA		00	X	1	X	X					
ALUSrcB		01	X	10	XX	XX					
ALUOp		00	X	00	XX	XX					
PCSrc		00	X	XX	XX	XX					
MemWr		0	0	0	0	0					

提纲

- 内容主要取材
  - 数字设计和计算机体系结构（第3章，第7章）
- 多周期数据通路控制信号分析
- 多周期控制器状态机构造
- 多周期性能分析

Review: Processor Performance

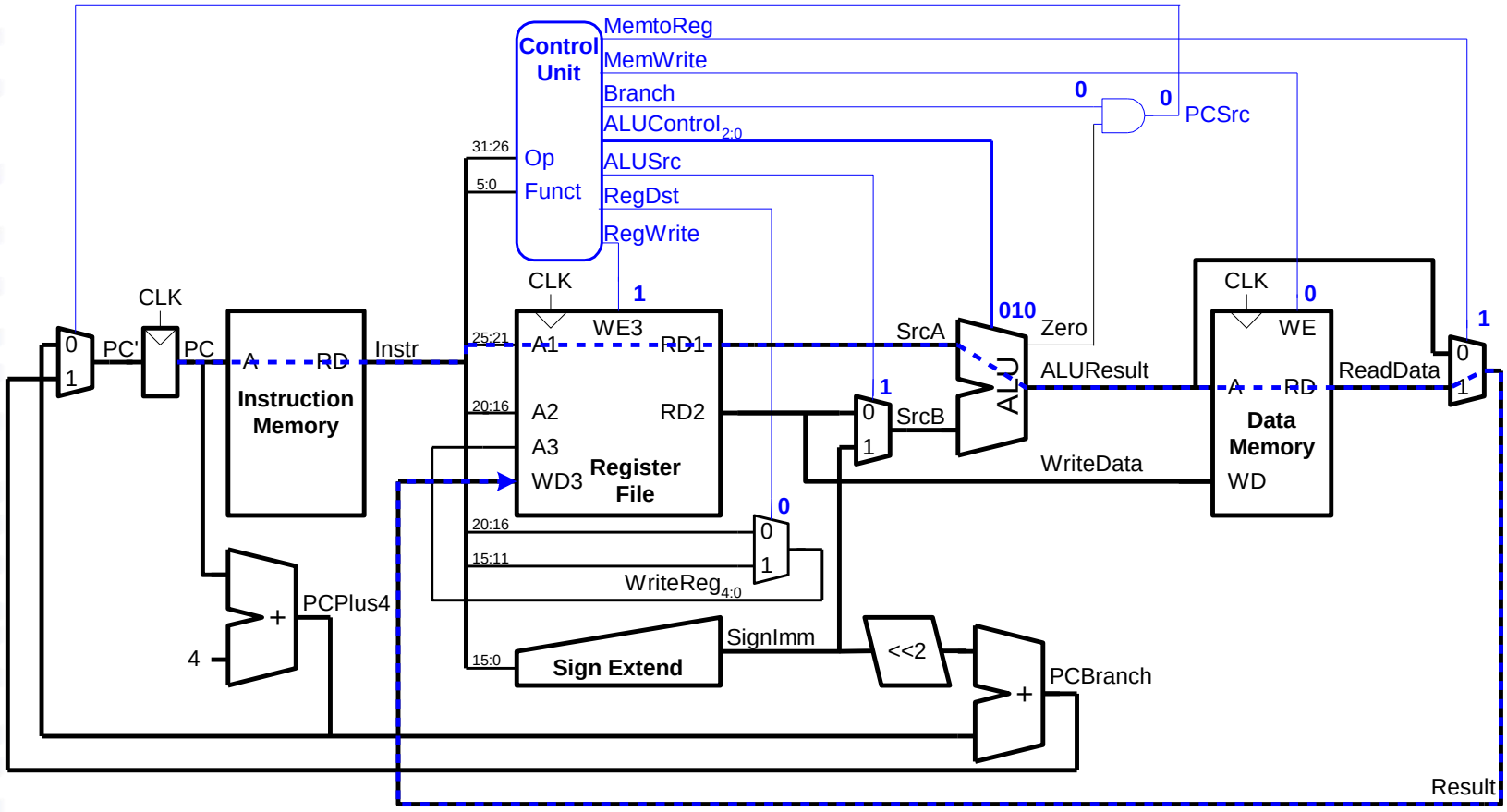
Program Execution Time

= (#instructions)(cycles/instruction)(seconds/cycle)

= # instructions x CPI x T_C

CPI: Cycle/instruction

Single-Cycle Performance



T_C limited by critical path (`lw`)

Single-Cycle Performance

- Single-cycle critical path:

$$T_c = t_{pcq\_PC} + t_{\text{mem}} + \max(t_{RF\text{read}}, t_{sext} + t_{\text{mux}}) + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$

- Typically, limiting paths are:
  - memory, ALU, register file
  - $T_c = t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{mux}} + t_{\text{ALU}} + t_{RF\text{setup}}$

Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

$T_c = ?$

Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

$$T_c = t_{pcq\_PC} + 2t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{RFsetup}$$
$$= [30 + 2(250) + 150 + 25 + 200 + 20] \text{ ps}$$
$$= 925 \text{ ps}$$

Single-Cycle Performance Example

Program with 100 billion instructions:

Execution Time = # instructions x CPI x T_C

$= (100 \times 10^9)(1)(925 \times 10^{-12}\ \text{s})$

$=$ **92.5 seconds**

Multicycle Processor Performance

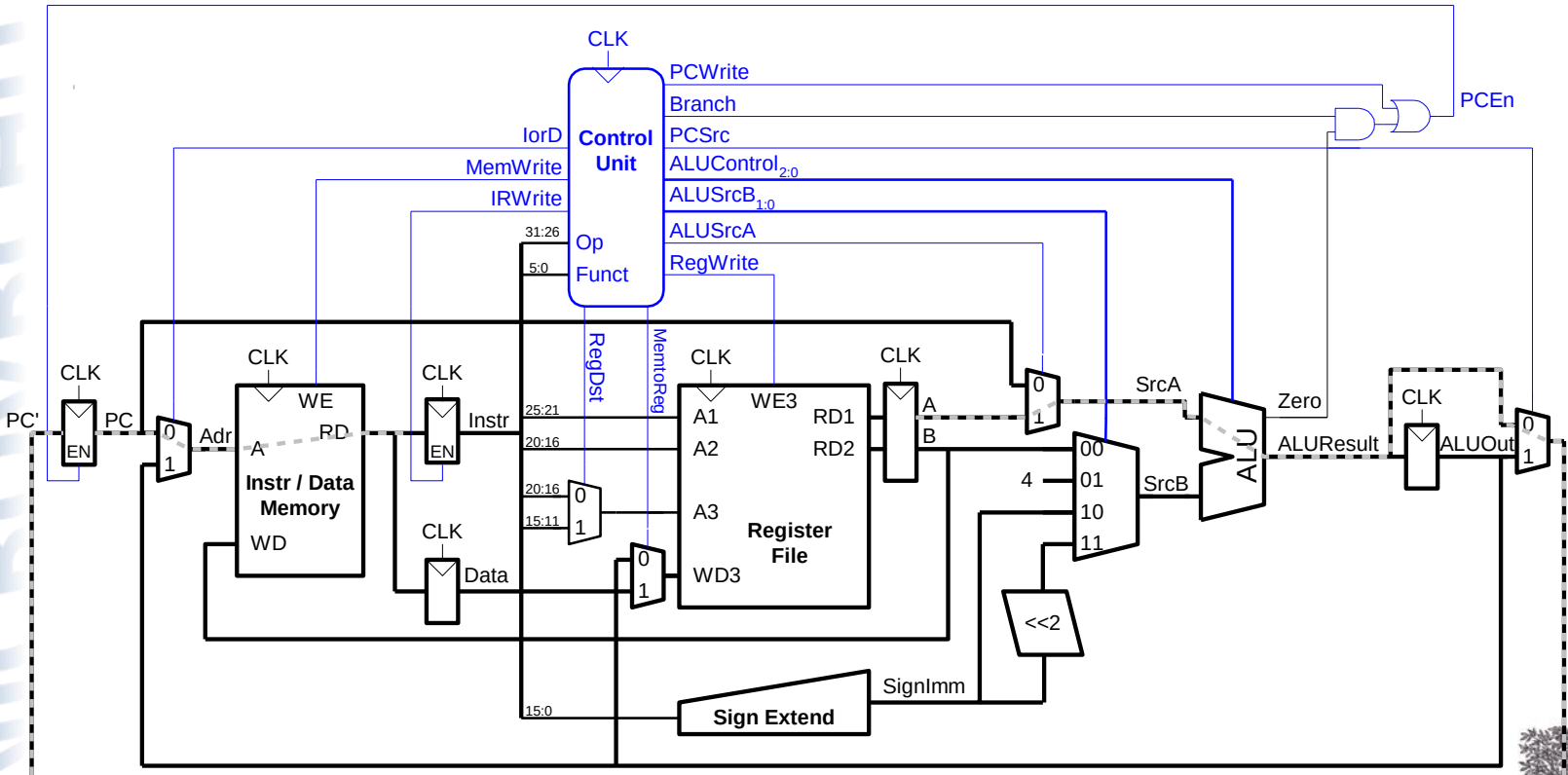
- Instructions take different number of cycles:
  - 3 cycles: `beq`, `j`
  - 4 cycles: R-Type, `sw`, `addi`
  - 5 cycles: `lw`
- CPI is weighted average
- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 11% branches
  - 2% jumps
  - 52% R-type

Average CPI = (0.11 + 0.2)(3) + (0.52 + 0.10)(4) + (0.25)(5) = **4.12**

Multicycle Processor Performance

Multicycle critical path:

$$T_c = t_{pcq} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup}$$



Multicycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

$T_c = ?$

Multicycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

$$T_c = t_{pcq\_PC} + t_{mux} + \max(t_{ALU} + t_{mux}, t_{mem}) + t_{setup}$$
$$= t_{pcq\_PC} + t_{mux} + t_{mem} + t_{setup}$$
$$= [30 + 25 + 250 + 20] \text{ ps}$$
$$= \mathbf{325 \text{ ps}}$$

Chapter 7 :: Topics

- For a program with 100 billion instructions executing on a multicycle MIPS processor
  - CPI = 4.12
  - T_c = 325 ps

Execution Time =

- For a program with 100 billion instructions executing on a multicycle MIPS processor
  - CPI = 4.12
  - T_c = 325 ps

Execution Time = (# instructions) × CPI × T_c

= $(100 \times 10^9)(4.12)(325 \times 10^{-12})$

= 133.9 seconds

- This is slower than the single-cycle processor (92.5 seconds). Why?

Multicycle Performance Example

Program with 100 billion instructions

Execution Time = (# instructions) × CPI × T_c

$= (100 \times 10^9)(4.12)(325 \times 10^{-12})$

$=$ **133.9 seconds**

This is **slower** than the single-cycle processor (92.5 seconds). Why?

– Not all steps same length

– Sequencing overhead for each step ($t_{pcq} + t_{\text{setup}} = 50$ ps)